廣東工業大學

《Python 数值分析》

课程报告

学　　院　　计算机学院

专　　业　　人工智能

班　　级　　1 班

学　　号　　3121005358

学生姓名　　欧炜标

授课教师　　胡晓敏

2023 年 5 月

摘要

本报告总结了数值分析课程的各章内容和实验分析。第一章介绍了数值分析的内容、特点、作用和历史。第二章重点介绍了误差相关实验与分析，强调了误差分析在数值运算中的重要性和复杂性，并提出了避免误差危害的方法和注意事项。第三章侧重于非线性方程的数值解法实验与分析，介绍了求解非线性方程的迭代法及其理论，并介绍了一些常用的算法和方法。第四章介绍了线性方程组的数值解法实验与分析，重点介绍了解线性方程组的直接方法和相关概念，如列主元 Gauss 消去法和矩阵的 LU 分解。第五章讨论了插值法的实验与分析，介绍了拉格朗日插值、牛顿插值和埃尔米特插值等方法，并强调了三次样条插值的重要性和应用。第六章探讨了最小二乘拟合的实验与分析，介绍了最小二乘法在函数逼近中的应用，并讨论了多项式拟合的问题。最后，第七章介绍了数值积分、微分和常微分方程的数值解法，重点介绍了逼近法在积分和微分计算中的应用，并通过实例讨论了欧拉方法和龙格-库塔方法的求解过程。第八章是全课程的内容总结以及个人的心得体会。

目录

1 绪论

1.1 数值分析的发展综述

数值分析处于现代数学研究的中心位置，在科学及工程领域具有无法替代的地位。

数值分析是研究连续问题的算法的科学。其中，最主要的概念就是算法和连续问题。首先，连续问题是从物理或者其它学科中抽象出来的复杂模型问题，一般是无穷维问题且几乎无法找到解析解。这些棘手的连续问题就自然成为数值分析的目标对象。其次，求解连续问题的算法的设计和分析是数值分析的核心内容，它们的目的是将连续的无穷维的问题离散化，得到一个离散的有限维的可解问题，进而得到近似解。如果没有数值分析，现代科学与工程应用研究将很快陷入停滞。

数值分析其实早在百余年前就已萌芽，其中，中国古代关于数值分析算法最典型的就是关于圆周率的计算——基于正多边形迭代的“割圆术”算法。

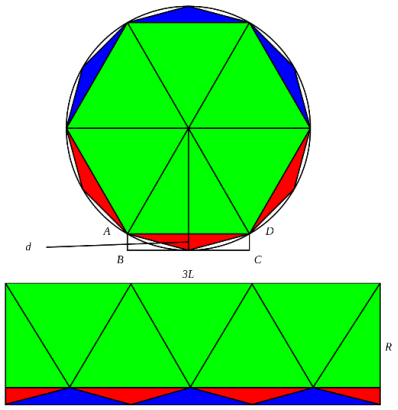


Figure 0-1：割圆术

而现代计算数学也叫做数值计算方法或数值分析，它是一门兼具基础性、应用性和边缘性的数学学科，主要内容包括代数程、线性代数方程组、微分方程的数值解法，函数的数值逼近题，矩阵特征值的求法，最优化计算问题，概率统计计算问题等等，还包括解的存在性、唯一性、收敛性和误差分析等理论问题。因此计算数学不能片面的理解为各种算法的简单罗列和堆积，现代计算数学从广义上可以分为数值算法和非数值算法。数值算法是研究解决数学问题并在计算机上使用的求数值近似解的方法，这类算法可以建立数学模型设计算法，这是通常所说的数值计算方法（简称计算方法），如求解方程的近似解等等。非数值算法是通过过程模型来描述算法，常用于事务管理领域，但非数值运算的种类繁多，要求各异，难以规范化，典型的非数值算法如排序、检索、表处理、判断、决策、形式逻辑演绎等等。

目前数值计算还由于多难题亟待解决。计算机通常处理的数据量很大，如果算法的时间复杂度很大，那么可能导致计算机无法完成任务。所以解决一个问题时选择的算法通常都需要是多项式级的复杂度，非多项式级的复杂度所需要的时

间太多，往往会超时。但是，并非所有的问题都能找到复杂度为多项式级别的算法。有些问题甚至根本不可能找到一个正确的算法来，这称之为“不可解问题”。以下简要介绍这类相关问题：

P 问题 (Polynomial Solvable)：如果一个判定问题是 P 问题，则这个问题存在一个多项式解法。即图灵机只需要多项式时间就可以得到答案, 既回答 yes 或者 no。

NP 问题(Nondeterminstic Polynomial Solvable)：如果一个判定问题是 NP 问题， 则这个问题的一个可能的解，可以在多项式时间内被验证是否正确。

P 属于 NP。 就是说，一个问题如果属于 P， 则一定属于 NP。反过来则不一定，7 大数学世纪难题之一就是问 P 是否等于 NP。

NPC, 即 NP 完全性问题。 任意一个 NP 问题都可规约到该问题，那么该问题称为 NP-complete（NPC）。 它是指 NP 问题中的最难的问题， 即还没有找到多项式解法，但多项式可验证。 而且只要一个 NPC 问题有多项式解法，其它所有 NP 问题都会有一个多项式解法。

NP-hard 是指所有还没有找到多项式解法的问题, 并没有限定属于 NP。所以 NP-hard 比 NPC 范围更大，也会更难。 NPC 是 NP-hard 和 NP 的交集.。NPC 问题都是 NP-Hard 问题。[1]

1.2 报告主要内容及结构

本报告将主要分为 6 章。

第一章是绪论，简要介绍数值分析方法的发展、数值分析的主要设计思想、数值分析的应用以及一些经典数值分析方法模型。

第二章是误差相关实验与分析。将介绍误差成因、误差的处理手段并就实例对其进行分析

第三章是非线性方法数值解法实验与分析。将由浅入深依次介绍二分法、不动点迭代法、Aitken 和 Steffensen 加速法、牛顿迭代法以及弦截法。

第四章是线性方程组的数值解法实验与分析。将中直接法开始到迭代法实现，依次介绍 Gauss 消去法、Gauss 列主元消去法、Gauss_Jordan 消去法、LU 分解法、Jacobi 迭代法以及 Gauss_Seidel 迭代法。

第五章是插值法的实验与分析。将从运动轨迹预测的实例出发，依次介绍拉格朗日多项式插值法、牛顿多项式插值法以及分段线性插值法。

第六章是最小二乘拟合的实验与分析。将从石油产量预测的实例出发介绍最小二乘拟合方法

第七章是数值积分、微分和常微分方程的数值解法。

第八章是总结与心得体会。

2 误差相关实验与分析

2.1 误差的成因与处理手段探讨

2.1.1 误差种类及成因

观察误差：由于人的鉴别力不同和外界条件等不可抗因素影响产生的误差。

模型误差：在建立数学模型过程中，要将复杂的现象抽象归结为数学模型，往往要忽略一些次要因素的影响，对问题作一些简化。因此数学模型和实际问题有一定的误差。

截断误差：由于实际运算只能完成有限项或有限步运算，因此要将有些需用极限或无穷过程进行的运算有限化，对无穷过程进行截断，这样产生的误差成为截断误差。

舍入误差：在数值计算过程中，由于计算工具的限制，我们往往对一些数进行四舍五入，只保留前几位数作为该数的近似值，这种由舍入产生的误差成为舍入误差。

2.1.2 以计算机为计算工具下对误差的预防和处理

1) 避免两个相近的数相减。
2) 防止大数吃小数。
3) 避免采用绝对值很小的数作为除数。
4) 简化运算步骤，减少运算次数。
5) 控制计算方法的误差传播，保证计算方法的稳定性。

2.2 学习数值计算方法的目的

数值分析是利用计算机求解实际问题的核心过程。虽然已有大量数值算法的软件包，但随着计算机的应用越来越广泛，计算问题越来越复杂，规模越来越大，现成的数值方法软件包不能满足特定需要，所以我们需要学习数值计算方法，了解算法设计的原理，以便更好地应用，以解决当下面临的难题。

2.3 综合实验：减少运算次数的实验

2.3.1 实验题目

比较不同算法求多项式的运算次数与用时。

设计 2 种不同的算法计算以下函数的值，分别测试 $x=0.1$，1，2。

$$f_n(x)=1+2x+3x^2+\cdots+100001x^{100000} \tag{1}$$

2.3.2 实验条件

计算机配置：Nitro AN515-57

CPU：11th Gen Inter(R) Core™ i7-11800H @ 2.30GHz(16 CPUs), ~2.3GHz

内存大小：16384MB RAM

操作系统：windows 10 家庭中文版 64 位（10.0，内部版本 19044）

2.3.3 算法介绍

算法 1：

对于多项式[4]

$$f_n(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n \tag{2}$$

分别计算 a_0, a_1x, a_2x^2, ... ,a_nx^n 的值，然后将其相加得到最终的结果 $f_n(x)$，在此过程中，一共需要执行 n 次加法以及 $\frac{n(n+1)}{2}$ 次乘法，比如：

当 $n=1$ 时， $f_1(x)=a_0+a_1x$，此时一共需要执行 1 次加法以及 1 次乘法;

当 $n=2$ 时， $f_1(x)=a_0+a_1x+a_2x^2$，此时一共需要执行 2 次加法以及 3 次乘法;

.....

当 $n=1000$ 时， $f_1(x)=a_0+a_1x+a_2x^2$，此时一共需要执行 1000 次加法以及 500500 次乘法。

.....

算法 2：

秦九韶法[5]，先算出每个系数对应的中间值，然后全部相加得到中间结果。其中加法部分的分析步骤和算法 1 类似。乘法部分采用复用的思路，大幅度减少了乘法的次数，其总体流程就是用先前系数乘法次方的结果来继续算后续系数乘法次方的结果，从而达到了复用的目的。

对于公式（2）的多项式，其迭代公式可变为

$$\begin{cases} S_n = a_n \\ S_k = xS_{k+1} + a_k, k = n-1, n-2\cdots, 1, 0 \\ f_n(x) = S_0 \end{cases} \quad (3)$$

通过利用公式（3）进行迭代运算，一共需要执行 n 次加法以及 n 次乘法即可求得结果。

2.3.4 实验结果及分析

表 1　两种解多项式算法性能对比图

x	算法	函数结果 f	乘法次数	加法次数	用时(秒)
0.1	算法 1	1.23456	5000050000	100000	0.038 秒
	算法 2	1.23456	100000	100000	0.017 秒
1	算法 1	5000150001	5000050000	100000	0.045 秒
	算法 2	5000150001	100000	100000	0.018 秒
2	算法 1	30109	5000050000	100000	9.4 秒
	算法 2	30109	100000	100000	0.27 秒

结果讨论：

经过分析，两种算法算出的结果是相同的，加法运算次数相同，乘法运算次数不同。由表 1 大致得出如下规律：

1）由于算法二的改进思路是用先前乘法的结果继续算后续的乘法结果，从而减少了乘法的次数，所以导致其乘法运算次数相比于算法一要低，因而用时更长由；

2）随着 x 的增大算法 2 的优势更明显，用时相比于算法 1 更具优势，这是因为算法 1 的乘法占比变得更大，因而算法 2 优势更明显；

3）无论是算法 1 还是算法 2，在同一个算法中，x 的取值越大，计算用时越长。这是因为 x 越大，乘法和加法的运算次数也会因此提高；

因此，算法 2 的算法性能要比算法 1 更具优势，并且由于乘法运算所需时间比较多，所以在计算过程中应要避免乘法的次数。

2.3.5 附录：源代码

【算法 1 源代码】

```
from time import *    #引入时间库
startT = time()       #记录起始时间
#  程序写在这里
x = 2
f = 1
countMul = 0 #统计乘法次数
countAdd = 0 #统计加法次数
```

```
for i in range(100000):
    f = f + (i + 2) * (x ** (i+1))    #函数 f
    countMul += (i+1)                 #统计加法次数
    countAdd += 1                     #统计乘法次数
endT = time() #记录结束时间
print("result = "，len(str(f)))
print("time = %.2g 秒\n" % (endT - startT))
print("乘法次数"，countMul)
print("加法次数"，countAdd)
```

【算法 2 源代码】

```
from time import * #引入时间库
startT = time() #记录起始时间
#  程序写在这里
x = 2
f = 1
powN = 100000   #最后一个数的幂次
aN = powN + 1   #最后一个系数值
countMul = 0      #统计乘法次数
countAdd = 0      #统计加法次数
startT = time()
S = aN              #函数值
for i in range(powN，0，-1):  #i 从 powN 开始到 1

    S = x * S + I                #迭代函数
    countAdd += 1              #统计加法次数
     countMul += 1             #统计乘法次数
endT = time() #记录结束时间
print("result = "，len(str(S)))
print("time = %.2g 秒\n" % (endT - startT))
print("乘法次数"，countMul)
print("加法次数"，countAdd)
```

3 非线性方程的数值解法实验与分析

非线性是实际问题中经常出现的，并且在科学与工程计算中的地位越来越重要，很多我们熟悉的线性模型都是在一定条件下由非线性问题简化得到的，为得到更符合实际的解答。往往需要直接研究非线性模型，从而产生非线性科学，它是 21 世纪科学技术发展的重要支柱。非线性问题的数学模型有无限维的如微分方程，也有有限维的。但要用计算机进行科学计算都要转化为非线性的单个方程或方程组的求解。从线性到非线性是一个质的变化，方程的性质有本质不同，求

解方法也有很大差别。本章将首先讨论单个方程求根，然后再简单介绍非线性方程组的数值解法。

3.1 求解非线性方程的二分法

3.1.1 实验题目

1. 通过二分法[6]求任意实函数方程$f(x)=0$在自变量$[a,b]$内或某一点附近的近似解，设置 LIMIT 作为绝对误差的分界线，不断迭代计算直到找到满足 LIMIT 的最近似解；

2. 深入理解误差线、精度的概率。

3.1.2 算法介绍

若 $f \in C[a,b]$，且 $f(a)\cdot f(b)<0$，则 f 在 (a,b) 上必有一根。每次迭代都使得$f(x)=0$ 的解所在的区间$[a_n,b_n]$缩小一半，直到 x 的值和真实解的误差满足所设精度才停止迭代。

3.1.3 实验结果及分析

表 2　二分法求解过程数据

迭代次数	下限 x_{low}	上限 x_{up}	$(x_{up}+x_{low})/2$	$f((x_{up}+x_{low})/2)$ 的正负性
0	1.3	1.5	1.4	< 0
1	1.4	1.5	1.45	> 0
2	1.4	1.45	1.425	> 0
3	1.4	1.425	1.4125	< 0
4	1.4125	1.425	1.41875	> 0
5	1.4125	1.41875	1.415625	> 0
6	1.4125	1.415625	1.414062	< 0
7	1.4140625	1.415625	1.414843	> 0
8	1.4140625	1.414843	1.414453	> 0
9	1.4140625	1.414453	1.414257	> 0
10	1.4140625	1.414257	1.414160	< 0
11	1.4141601	1.414257	1.414208	< 0
12	1.4142089	1.414257	1.414233	> 0
13	1.4142089	1.414233	1.414221	> 0
14	1.4142089	1.414221	1.414215	> 0

15	1.4142089	1.414215	1.414212	< 0
16	1.4142120	1.414215	1.414213	< 0
17	1.4142135	1.414215	1.414214	> 0
18	1.4142135	1.414214	1.414213	> 0
19	1.4142135	1.414213	1.414213	> 0
20	1.4142135	1.414213	1.414213	> 0
21	1.4142135	1.414213	1.414213	> 0
22	1.4142135	1.414213	1.414213	> 0
23	1.4142135	1.414213	1.414213	> 0
24	1.4142135	1.414213	1.414213	> 0
25	1.4142135	1.414213	1.414213	> 0
26	1.4142135	1.414213	1.414213	> 0
27	1.4142135	1.414213	1.414213	> 0
28	1.4142135	1.414213	1.414213	> 0
29	1.4142135	1.414213	1.414213	< 0
30	1.4142135	1.414213	1.414213	< 0
31	1.4142135	1.414213	1.414213	< 0

结果讨论：

通过**表 2** 可知，随着迭代次数的不断增加，上下限的差距越来越近，得到的结果也越来越接近精确解。通过设置更加严格的 LIMIT 精度值，二分法得到的近似解的结果也越来越接近精确解，进而达到提升结果精度的效果。

3.1.4 附录：源代码

```
# 使用 Numpy 科学计算程序包
import numpy as np
# 用 plt 输入 matplotlib 的 pyploy
import matplotlib.pyplot as plt
# 设定 x 轴的范围和精度，生成一组等间距的数据
x = np.linspace(-5，5，100) # 获得 x 坐标数组
# step = 0.01       # 画图点之间的步长距离
# x = np.arange(-5，5+step，step)     # 获得 x 坐标数组
y = x * x - 2     # 函数 y = f(x)值
plt.figure()      # 创建 figure 对象
# 设定为 1 个图表显示
# plt.subplot(1，1，1)
# 线型图
plt.plot(x，y，label = 'line')     # 绘制关于 x 和 y 的折线图
plt.plot([-4，4]，[0，0])     # 绘制点(-4，0)到点(4，0)的直线
# 绘制符合要求的方程解
plt.plot(2 ** 0.5，0，marker = 'o'，markersize = 10)
```

```
import csv
a = 2     # f(x) = x * x - a
LIMIT = 1e-20
#  方程函数 f()定义
def f(x):
      '''函数值的计算'''
      return x * x - a
# f()函数结束

# ----主执行部分----
      #  初始设置
xlow = float(input("请输入 x 值下限:"))
xup = float(input("请输入 x 值上限:"))
#  循环处理
iter = 0                              #  迭代计数
while (xup - xlow) * (xup - xlow) > LIMIT:#  满足终止条件前循环
      mid = (xup + xlow) / 2     #  计算新的中值点
      iter += 1                        #  迭代计数加 1
      if(f(mid) > 0):                #  中点函数值为正
            print("大于，")
            xup = mid                 #  更新 xup
      if(f(mid) < 0):                 #  中点函数值为负值
            print("小于，")
            xlow = mid                #  更新 xlow
      print("{:.15g}    ，      {:.15g}    ，     {:.15g}".format(iter，xlow，xup))
```

附加题：

对于大量的输出数据，有什么便捷的方法把列表数据输出，方便贴到 word 文档呢？或者直接把数据矩阵输出到.txt 文件中呢？

答：

在 while 循环中的 print 输出各个变量时，要用逗号分隔开不同的变量，然后运行程序。程序运行完后，复制控制台输出的所有结果，然后新建一个 excel 文件并将所复制结果粘贴到文档中，保存后将 excel 文件的文件类型后缀名改为.csv。此时开打改 excel 文件后，便可以把列表数据方便地粘贴到 word 文档中。

3.2 Python 绘图模拟非线性方程求解过程

3.2.1 实验题目

通过动态图显示求解迭代过程

对不动点迭代法 fixPointFig.py 进行修改，使得

1）图形坐标根据迭代范围缩小自动调整；

2）在图像上显示迭代序号，和对应迭代得到的值

3.2.2 算法介绍

1 中文显示：

为了能够正常显示中文，需要在代码开头添加如下代码：

```
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

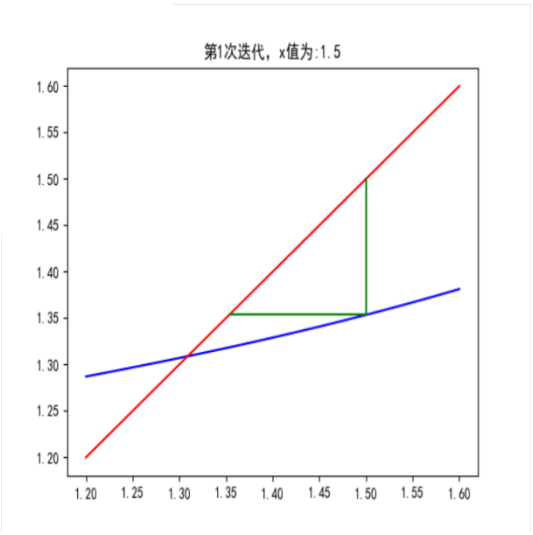
2 坐标调整与过程显示：

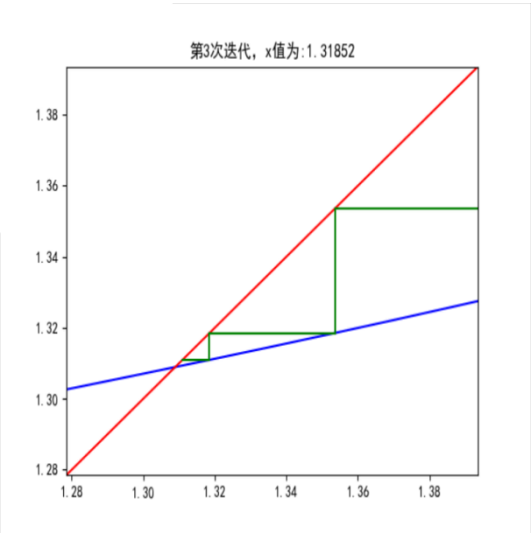
坐标调整与过程显示的关键代码为 plt.xlim 和 plt.ylim，由于每次迭代均会使得 x 和 y 变化的范围逐渐缩小，故需要用到 xlim 和 ylim 来缩小 x 和 y 轴的标度。

3 实现动态效果：

首先使用 plt.figure 创建画布，然后在一个循环中通过 plt.pause 控制绘图的等待时间，在每个等待时间内仅使用一次 plt.plot 在画布上绘制图片，从而达到动态显示效果。

3.2.3 实验结果及分析





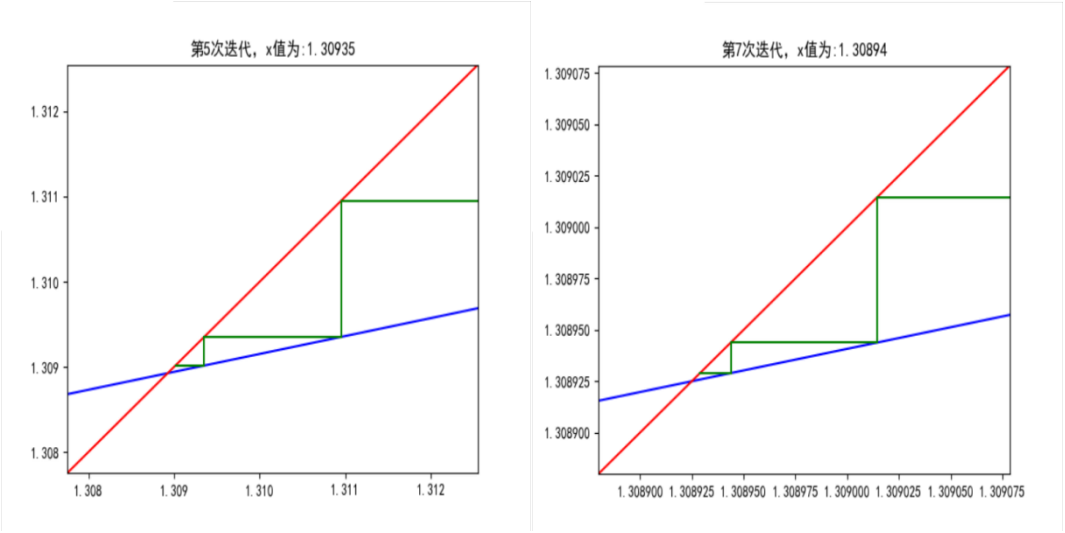


图 3.2　不动点迭代法动态显示过程（节选）

结果讨论：

设置近似解的精度为 10-5，如图 3.1 为不动点迭代法的动态执行过程部分截图，其中红色线为 $y=\left((-58\times x-3)\ /\ \left(7\times x^{3}\ -\ 13\times x^{2}-21\times x-12\right)\right)^{\frac{1}{2}}$，蓝色线为 $y=x$,绿线为不动点迭代法的迭代过程。从图上可见随着迭代次数的增加，x 和 y 轴的上下限范围也不断得缩小，x 的值不断接近符合精度的近似解，绿色的终点也不断接近红线和蓝线的交点（交点的横坐标即为方程的准确解 x），直到迭代第 8 次后近似解 x 的值满足精度 10-5。

3.2.4 附录：源代码

```
# -*- coding: utf-8 -*-
from matplotlib import pyplot as plt
import numpy as np
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

def fixpt(f, x, epsilon=1.0E-5, N=500, store=False):
    y = f(x)
    n = 0
    if store:
        Values = [(x, y)]
    while abs(y-x) >= epsilon and n < N:
        x = f(x)
        n += 1
        y = f(x)
        if store:
            Values.append((x, y))
```

```
    if store:
        return y, Values
    else:
        if n >= N:
            return "No fixed point for given start value"
        else:
            return x, n, y
# define f
def f(x):
    return ((-58 * x - 3) / (7 * x ** 3 - 13 * x ** 2 - 21 * x - 12)) ** (1 / 2)
# find fixed point
res, points = fixpt(f, 1.5, store = True)
# create mesh for plots
xx = np.arange(1.2, 1.6, 1e-5)
#plot function and identity
plt.plot(xx, f(xx), 'b')
plt.plot(xx, xx, 'r')

# plot lines
a = 0.2
i = 1
for x, y in points:
    plt.title(f"第{i}次迭代，x 值为:{x}")
    i = i + 1
    plt.plot([x, x], [x, y], 'g')
    plt.pause(0.3)
    plt.plot([x, y], [y, y], 'g')
    plt.pause(0.3)
    plt.xlim(y-a,x+a)
    plt.ylim(y-a,x+a)
    a = a / 5
# show result
    # plt.show()
```

3.3Aitken 和 Steffensen 方法加速求根

3.3.1 实验题目

编写 Aitken[7]和 Steffensen[8]方法加速例 2.2 的 5 种迭代格式的结果，写成分析报告（计算结果输出到文件，精度要求为 1e-5）。

通过计算机实现 Aitken 和 Steffensen 来求解非线性方程的解;

深入理解 Aitken 和 Steffensen 的几何意义;

深入理解 Aitken 和 Steffensen 的收敛条件及收敛阶数;

通过编程来设计多种迭代格式，并对比不同的不动点迭代格式的收敛速度，最后将结果输出到文件保存下来

3.3.2 算法介绍

采用一般迭代加速公式，经常会遇到$\varphi'(x)$不容易估计等难题，为避免该问题，可采用以下方法进行迭代加速：

假设{xk}是方程 x=φ(x)的近似跟序列，并且具有线性收敛速度。设$\bar{x}$k+1为近似值 xk 经过一次迭代后的结果，即

$$\bar{x}_{k+1}=\varphi(x_k),$$

又设 x*为迭代方程的根，即

$$x^*=\varphi(x^*),$$

由微分中值定理可得

$$x^*-\bar{x}_{k+1}=\varphi'(\zeta)(x^*-x_k),$$

其中ζ为 x*与 xk 之间的某点。假如$\varphi'(\zeta)$在求根范围内改变不大，则可近似地取某个定值 L，即

$$x^*-\bar{\bar{x}}_{k+1}\approx L(x^*-x_k) \tag{3.1}$$

再将迭代值$\bar{x}_{k+1}$用迭代公式校正一次得

$$\bar{x}_{k+1}=\varphi(\bar{x}_{k+1})$$

同样地，有

$$x^*-\bar{\bar{x}}_{k+1}\approx L(x^*-x_{k+1}) \tag{3.2}$$

式(3.1)和(3.2)两式相除得

$$\frac{x^*-\bar{\bar{x}}_{k+1}}{x^*-\bar{x}_{k+1}}\approx\frac{x^*-\bar{x}_{k+1}}{x^*-x_k}$$

整理得

$$x_{k+1}=\bar{\bar{x}}_{k+1}-\frac{(\bar{\bar{x}}_{k+1}-\bar{x}_{k+1})^2}{\bar{\bar{x}}_{k+1}-2\bar{x}_{k+1}+x_k} \tag{3.3}$$

则 xk+1 是比 $\bar{x}_{k+1}$ 和 $\bar{\bar{x}}_{k+1}$ 更好的近似值。综上得出 Atiken 迭代加速公式[2]如下：

$$\bar{x}_{k+1} = \varphi(x_{k+1}),$$

$$\bar{\bar{x}}_{k+1} = \varphi(\bar{x}_{k+1}),$$

$$x_{k+1} = \bar{\bar{x}}_{k+1} - \frac{(\bar{\bar{x}}_{k+1} - \bar{x}_{k+1})^2}{\bar{\bar{x}}_{k+1} - 2\bar{x}_{k+1} + x_k}$$

3.3.3 实验结果及分析

表 3.2　Aitken 方法迭代过程数据

迭代次数 n	(1)	(2)	(3)	(4)	(5)
0	1.5	1.5	1.5	1.5	1.5
1	1.50002	1.02102	-2.10438	1.30825	1.30751
2	1.50004	0.307844	2.76904	1.30889	1.30887
3	1.50007	-15.7335	2.76685	1.30892	1.30892
4	1.50009	4.32142			
5	1.50011	3.2787			
6	1.50013	2.99777			
7	1.50015	2.88398			
8	1.50018	2.82991			
9	1.50020	2.80201			
10	……	2.78689			
11		2.77845			
12		2.77363			
13		2.77085			
14		2.76922			
15		2.76826			
16		2.76769			
17		2.76735			
18		2.76715			
19		2.76703			
20		2.76696			
21		2.76691			
22		2.76689			

表 3.3 Steffensen 加速迭代法求解过程数据

n	（1）	（2）	（3）	（4）	（5）
0	1.5	1.5	1.5	1.5	1.5
1	1.50002	1.02102	-2.10438	1.30825	1.30751
2	1.50004	1.38449	2.65482	1.30892	1.30892
3	1.50007	1.94712	2.76673	1.30892	1.30892
4	1.50009	3.47192	2.76685		
5	1.50011	2.81754	2.76685		
6	1.50013	2.76729			
7	1.50015	2.76685			
8	1.50018	2.76685			
9	1.50020				
10	……				

结果讨论：

取初值 x_0 = 1.5，列出迭代过程如表 3.2 和表 3.3 所示，停止条件 $|x_n - x_{n-1}| < 10^{-5}$。有表 3.2 可以看出，$\varphi_1(x)$、$\varphi_2(x)$ 迭代了多个轮次后，依旧无法得到满足停止条件得准确解 1.30892；而 $\varphi_4(x)$ 和 $\varphi_5(x)$ 仅仅迭代了 3 个轮次，就已经得到了满足精度得准确解，这是由于 $\varphi_4(x)$ 和 $\varphi_5(x)$ 这两个迭代公式满足压缩定理可以收敛，而 $\varphi_1(x)$、$\varphi_2(x)$ 和 $\varphi_3(x)$ 并不满足，所以其无法收敛。

通过表 3.3 得数据能得出 Steffensen 方法具有以下特点：即使 φ 发散，得到的数列仍有可能收敛到不动点。它在一阶收敛到不动点的数列上，经过一次迭代能够实现二阶收敛。对于原本不收敛的数列，Steffensen 方法有时可以改进为二阶收敛。然而，如果迭代数列本身就是超线性收敛（即大于一阶），则改用 Steffensen 方法的意义并不大。因此，Steffensen 方法通常用于改进一阶收敛方法。

3.3.4 源代码

```
import math
import numpy as np
import matplotlib.pyplot as plt
from sympy import *

#-------全局变量定义-------
```

```
# LIMIT = 1e-5
# xlow = 1
# xup = 2
LIMIT = 1e-10
xlow = 1.3
xup = 2.5

#---------函数定义--------
def F(x):            #方程函数 f()定义
    # return 7.0 * math.pow(x,5) - 13.0 * math.pow(x,4) - 21 * math.pow(x,3) - 12 *
math.pow(x,2) + 58 * x + 3
    return x * x - 2

#待求解函数的一阶导数
def df(x):
    # y=35.0*x*x*x*x*x-52.0*x*x*x*x-63.0*x*x*x-24.0*x+58
    y = 2 * x
    return y

def pF1(x):        #不动点迭代法函数 1
    return 7 * math.pow(x,5) - 13 * math.pow(x,4) - 21 * math.pow(x,3) - 12 * math.pow(x,2) +
59 * x + 3

def pF2(x):        #不动点迭代法函数 2
    return math.pow((13 * math.pow(x,4) + 21 * math.pow(x,3) + 12 * x**2 - 58 * 2 - 3),1/5)

def pF3(x):        #不动点迭代法函数 3
    return (13 + 21 / x + 12 / math.pow(x,2) - 58 / math.pow(x,3) - 3 / math.pow(x,4)) / 7

def pF4(x):        #不动点迭代法函数 4
    return math.pow((12.0 * math.pow(x,2) - 58.0 * x - 3) / (7.0 * math.pow(x,2) - 13.0 * x -
21),1/3)

def pF5(x):        #不动点迭代法函数 5
    return math.pow((-58 * x - 3) / (7.0 * math.pow(x,3) - 13.0 * math.pow(x,2) - 21 * x -
12),1/2)

def aitF(pF,x1,x0):        #Aitken 埃特金加速迭代函数 pF
    x2 = pF(x1)
    return x0 - math.pow(x1-x0,2) / (x2 - 2 * x1 + x0)

def steF(pF,x):        #Steffensen 斯提芬森加速迭代函数 pF
```

```
    return x - math.pow(pF(x) - x,2) / (pF(pF(x)) - 2 * pF(x) + x)

#------------------------------算法代码------------------------------
def aitken(f,LIMIT):
    fileptr = open("aitken.txt", "w", encoding='utf-8')
    iter = 0    # 迭代计数
    x0 = input("请输入 aitken 迭代法的初值：x0 = ")
    x0 = float(x0)
    x1 = f(x0)
    print("{:.15g}\t{:.15g}\n".format(iter, np.around( x0,5)))
    fileptr.writelines("{:.15g}\t{:.15g}\n".format(iter, np.around( x0,5)))
    iter += 1

    while(1):
        if ((np.fabs(x1) <= 1) & (float(np.fabs(x1 - x0)) < LIMIT)):
            break
        elif ((np.fabs(x1) >= 1) & (float(np.fabs(x1 - x0) / np.fabs(x1)) < LIMIT)):
            break

        x2 = aitF(f,x1,x0)
        print("{:.15g}\t{:.15g}\n".format(iter, np.around( x2,5)))
        fileptr.writelines("{:.15g}\t{:.15g}\n".format(iter, np.around( x2,5)))

        x0 = x1
        x1 = x2
        iter += 1

    fileptr.close()

def steffensen(f,LIMIT):
    fileptr = open("steffensen.txt", "w", encoding='utf-8')
    iter = 0    # 迭代次数
    x0 = input("请输入 steffensen 迭代法的初值：x0 = ")
    x0 = float(x0)
    print("{:.15g}\t{:.15g}\n".format(iter, np.around( x0,5)))
    fileptr.writelines("{:.15g}\t{:.15g}\n".format(iter, np.around( x0,5)))
    iter += 1

    while(1):
        x1 = steF(f,x0)
        print("{:.15g}\t{:.15g}\n".format(iter, np.around( x1,5)))
        fileptr.writelines("{:.15g}\t{:.15g}\n".format(iter, np.around( x1,5)))
```

```
            if ((np.fabs(x1) <= 1) & (float(np.fabs(x1 - x0)) < LIMIT)):
                break
            elif ((np.fabs(x1) >= 1) & (float(np.fabs(x1 - x0) / np.fabs(x1)) < LIMIT)):
                break
            x0 = x1
            iter += 1

    fileptr.close()
def main():
    if xlow >= xup:
        print("Error!xlow %g should be smaller than xup %g"%(xlow,xup))
        return

    #ait 迭代法
    # aitken(pF2,LIMIT)

    #ste 迭代法
     # steffensen(pF5,LIMIT)
main()          #执行主程序
```

3.4 综合实验： 多种方法对比

3.4.1 实验题目

采用二分法、不动点迭代法、加速收敛方法、牛顿法、弦截法对例 2.1 方程进行求解。列出求解数据表格，并分析。（注意初始值的设置尽量相同）

本次作业求解方程如下：

$$7x^5-13x^4-21x^3-12x^2+58x+3=0, x\in[1,2]$$

3.4.2 算法介绍

二分法：

原理：若 $f\in C[a,b]$，且 $f(a)\cdot f(b)<0$，则 f 在 (a,b) 上必有一根。

第一步，取 $a=a_0$， $b=b_0$，中点 $x_0=\dfrac{a_0+b_0}{2}$，如果 $f(x_0)=0$，那么 $p=x_0$，即 x_0 为根，算法停止。否则，如果 $f(a_0)f(x_0)<0$，取 $a_1=a_0$， $b_1=x_0$；如果 $f(x_0)f(b_0)<0$，取 $a_1=x_0$， $b_1=b_0$，此时方程的有根区间缩小为 $[a_1,b_1]$。

第二步，区间$[a_1,b_1]$的中点为$x_1=\frac{a_1+b_1}{2}$，如果$f(x_1)=0$，那么$p=x_1$，即x_1为根，算法停止。否则，如果$f(a_1)f(x_1)<0$，取$a_2=a_1$，$b_2=x_1$；如果$f(x_1)f(b_1)<0$，取$a_2=x_1$，$b_2=b_1$，此时方程的有根区间缩小为$[a_2,b_2]$。

重复上述步骤，不断缩小有根区间，直至找到满足精度的解。

不动点迭代法：

不动点法(fixed point method)是解方程的一种一般方法，对研究方程解的存在性、唯一性和具体计算有重要的理论与实用价值。数学中的各种方程，诸如代数方程、微分方程和积分方程等等，均可改写成$x=\varphi(x)$的形式，其中x是某个适当的空间X中的点，φ是从X到X的一个映射，把点x变成点$\varphi(x)$。于是，方程的解就相当于映射φ在空间X中的不动点。这一方法把解方程转化为求某个映射的不动点，故而得此名。其收敛阶一般为1。

特点：

A. 构造在区间[*a*, *b*]满足压缩映射的迭代函数；

B. 选择迭代初值，迭代求解；

C.收敛阶一般为1。

假设求解$f(x)=0$，由先验知识可知求$f(x)$的根相当于求$x=\varphi(x)$*的不动点*。从一个初值x_0出发，计算$x_1=\varphi(x_0)$，$x_2=\varphi(x_1)$，...，$x_n=\varphi(x_{n-1})$，若存在p使得$\lim_{n\to\infty}x_{n+1}=\lim_{n\to\infty}\varphi(x_n)$可知$p=\varphi(x_0)$，即$p$是$\varphi$的不动点，也就是f的根。由于将方程$f(x)=0$转换为$x=\varphi(x)$有不同的方式，因此对方程$f(x)=0$来说，$\varphi(x)$也不是唯一的。

例如本次作业求解方程如下：

$$7x^5-13x^4-21x^3-12x^2+58x+3=0,x\in[1,2] \tag{3.4}$$

通过把要求解的方程写成迭代形式，可以给出如下5种不同的迭代形式，其中前三种都不满足压缩映射要求，后两种格式满足区间[1,2]压缩映射要求。需要注意的是，不同的映射方式将会影响方程的收敛速度。

$$x=\varphi_1(x)=7x^5-13x^4-21x^3-12x^2+59x+3 \tag{3.5}$$

$$x=\varphi_2(x)=\left(\frac{13x^4+21x^3+12x^2-58x-3}{7}\right)^{\frac{1}{5}} \tag{3.6}$$

$$x=\varphi_3(x)=\frac{13+\frac{21}{x}+\frac{12}{x^2}-\frac{58}{x^3}-\frac{3}{x^4}}{7} \tag{3.7}$$

$$x=\varphi_4(x)=\left(\frac{12x^2-58x-3}{7x^2-13x-21}\right)^{\frac{1}{3}} \tag{3.8}$$

$$x=\varphi_5(x)=\left(\frac{-58x-3}{7x^3-13x^2-21x-12}\right)^{\frac{1}{2}} \tag{3.9}$$

加速收敛方法：

Aitken：

该方法用于加速线性收敛数列的收敛速度，对于一个收敛到 p 的数列$\{x_n\}$，如果其收敛阶为 1，那么 $\lim\limits_{n\to\infty}\frac{|x_{n+1}-p|}{|x_n-p|}=C$ 且常数 $0<C<1$，可以认为当 n 趋近于无穷时有 $\frac{|x_n-p|}{|x_{n-1}-p|}\approx\frac{|x_{n-1}-p|}{|x_{n-2}-p|}$，整理得迭代数列 $\hat{x}_n=x_n-\frac{(x_n-x_{n-1})^2}{x_n-2\times x_{n-1}+x_{n-2}}$ 称为$\{x_n\}$得 Aitken 数列。使用 Aitken 方法加速，只要有相邻三项。Aitken 方法可对一阶收敛的数列加速到二阶收敛，所以其收敛阶一般大于 1。特点：A. 构造在区间[*a, b*]满足压缩映射的迭代函数 B. 选择迭代初值，采用 Aitken 数列公式迭代求解。

Steffensen：

加速收敛迭代格式的思想和 Aitken 加速方法用于一般迭代法类似。在 Aitken 基础上经简单的算术运算，便可得到 Steffensen 得迭代公式为 $\hat{x}_n=x_{n-1}-\frac{(\phi(x_{n-1})-x_{n-1})^2}{\phi(\phi(x_{n-1})-2\times\phi(x_{n-1})+x_{n-1})}$。相比于 Aitken，通过 Steffensen 迭代序列来迭代计算结果即可获得更精确的近似解。其收敛阶一般为 2 或以上。特点：A. 把方程改写成等价形式 *x=f(x)* B.选择迭代初值，采用 Steffensen 数列公式迭代求解 C.收敛阶一般为 2 或以上。[2]

牛顿法：

牛顿迭代法（Newton's method）又称为牛顿-拉夫逊（拉弗森）方法（Newton-Raphson method），它是牛顿在 17 世纪提出的一种在实数域和复数域上近似求解

方程的方法。

其原理是将非线性方程线性化——泰勒展开取 $x_0 \approx p$，将 $f(x)$在x_n做一阶泰勒展开，然后去掉 2 阶及 2 阶以上项，则得到迭代公式 $x_{n+1} \approx x_n - \frac{f(x_n)}{f`(x_n)}$。

需要注意的是，牛顿切线法的收敛性依赖于x_0的选取。特点：A. 为了保证收敛，对初值的选取有要求 B. 需要求导数 C.在单根附近至少 2 阶收敛，但在重根附近线性收敛。

弦截法：

Newton 迭代法有一个较强的要求是 $f`(x_n)$ 存在，且不为零。因此，用弦的斜率近似的替代 $f`(x_n)$。设 $f(x_n)$ 在$[a,b]$上有唯一的零点 x^*，取 $x_0 = a$， $x_1 = b$，则过 $P_0(x_0, f(x_0))$ 及 $P_1(x_1, f(x_1))$ 得弦得方程： $y = f(x_1) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} \times (x - x_1)$。

定端点弦截法：

又称单点割线法、线性插值二分法，给定x_1和x_0的值，对于 n=1，2...有 $x_{n+1} = x_n + \frac{x_n - x_0}{f(x_n) - f(x_0)} \times f(x_n)$。

动端点弦截法：

给定x_n和x_{n-1}的值，对于 $n = 1, 2...$有 $x_{n+1} = x_n + \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \times f(x_n)$。特点：A.为例保证手里，对初值的选取有要求。B. 收敛阶一般为 1.618。

3.4.3 实验结果及分析

表 4　迭代法求解非线性方程的迭代速度比较表

迭代	二分法	不动点 φ_4	不动点 φ_5	Aitken 加速		Steffensen 加速		牛顿法	定截弦法	动截弦法
				φ_4	φ_5	φ_4	φ_5			
0	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	2	2
1	1.25	1.36538	1.35354	1.30825	1.30751	1.30825	1.30751	1.3261	1	1
2	1.375	1.32528	1.31852	1.30889	1.30887	1.30892	1.30892	1.30914	1.21359	1.21359
3	1.3125	1.31364	1.31095	1.30892	1.30892	1.30892	1.30892	1.30892	1.28755	1.34572

4	1.28125	1.31028	1.30935	1.30892	1.30474	1.30612
5	1.29688	1.30932	1.30901		1.30813	1.30885
6	1.30469	1.30904	1.30894		1.30878	1.30893
7	1.30859	1.30896	1.30893		1.3089	1.30892
8	1.31055	1.30893			1.30892	
9	1.30957				1.30892	
10	1.30908					
11	1.30884					
12	1.30896					
13	1.3089					
14	1.30893					
15	1.30891					
16	1.30892					
17	1.30893					

用不同的非线性方程迭代法求解方程在区间[1.3, 1.5]上的根，设定停止条件为$|x_n - x_{n-1}| < 10^{-10}$，经分析得如下结论：

1）由于二分法是线性收敛，不动点迭代法收敛阶一般为 1，Aitken 迭代法收敛阶一般大于 1，Steffensen 迭代法收敛阶一般为 2 或以上，牛顿法收敛阶至少为 2，弦截法收敛阶一般为 1.618，所以从**表 4** 中可以看出在相同条件下牛顿法和 Steffense 的收敛速度最快。

2）由**表 4** 中也可知道定端点法要比动端点法的迭代速度要稍微慢一些，这是因为定端点法每次只更新一个点，而动端点每次会迭代更新两个点。

3）二分法虽然简单易用使用性强，使用性强，不需要考虑初值的选取，只需考虑连续函数在区间两端点异号的条件，但是收敛速度慢。

4）迭代法理论丰富、算法简单，构造出合适的迭代函数即可加快收敛速度。但不易找到收敛最快的迭代函数和收敛到解的迭代函数。

5）对于不动点迭代法、Aitken 和 Steffensen 迭代法，构造出合适的迭代函数可以加快收敛速度，但是实际上并不容易构造能收敛到解的迭代函数，比如迭代函数Φ_4并不满足压缩定理，采用不动点法并能得到满足精度的近似解；而迭代函数Φ_5满足压缩定理，所以此时三种方法均能得到满足精度的近似解，并且Φ_5的迭代收敛速度普遍要比Φ_4的快。

6）对于牛顿法和弦截法，为保证收敛，其均对初值的选取有要求，比如**表 4** 中均选取初值为 1.4 时能得到收敛解。通过实验发现，当初值为其他值时确实并不一定能保证收敛。

7）弦截法虽然迭代次数比牛顿法中多，对初值的选取仍有要求，但是用差商取代了牛顿法的导数运算。

3.4.4 附录：源代码

```
import math
import numpy as np
import matplotlib.pyplot as plt
from sympy import *

#-------全局变量定义-------
# LIMIT = 1e-5
# xlow = 1
# xup = 2
LIMIT = 1e-10
xlow = 1.3
xup = 2.5

#---------函数定义--------
def F(x):            #方程函数 f()定义
    # return 7.0 * math.pow(x,5) - 13.0 * math.pow(x,4) - 21 * math.pow(x,3) - 12 * math.pow(x,2)
+ 58 * x + 3
    return x * x - 2

#待求解函数的一阶导数
def df(x):
    # y=35.0*x*x*x*x-52.0*x*x*x-63.0*x*x-24.0*x+58
    y = 2 * x
    return y

def pF1(x):        #不动点迭代法函数 1
    return 7 * math.pow(x,5) - 13 * math.pow(x,4) - 21 * math.pow(x,3) - 12 * math.pow(x,2) +
59 * x + 3

def pF2(x):        #不动点迭代法函数 2
    return math.pow((13 * math.pow(x,4) + 21 * math.pow(x,3) + 12 * x**2 - 58 * 2 - 3),1/5)
```

```
def pF3(x):        #不动点迭代法函数 3
    return (13 + 21 / x + 12 / math.pow(x,2) - 58 / math.pow(x,3) - 3 / math.pow(x,4)) / 7

def pF4(x):        #不动点迭代法函数 4
    return math.pow((12.0 * math.pow(x,2) - 58.0 * x - 3) / (7.0 * math.pow(x,2) - 13.0 * x -
21),1/3)

def pF5(x):        #不动点迭代法函数 5
    return math.pow((-58 * x - 3) / (7.0 * math.pow(x,3) - 13.0 * math.pow(x,2) - 21 * x - 12),1/2)

def aitF(pF,x1,x0):        #Aitken 埃特金加速迭代函数 pF
    x2 = pF(x1)
    return x0 - math.pow(x1-x0,2) / (x2 - 2 * x1 + x0)

def steF(pF,x):        #Steffensen 斯提芬森加速迭代函数 pF
    return x - math.pow(pF(x) - x,2) / (pF(pF(x)) - 2 * pF(x) + x)

#------------------------------算法代码------------------------------
def aitken(f,LIMIT):
    fileptr = open("aitken.txt", "w", encoding='utf-8')
    iter = 0   # 迭代计数
    x0 = input("请输入 aitken 迭代法的初值：x0 = ")
    x0 = float(x0)
    x1 = f(x0)
    print("{:.15g}\t{:.15g}\n".format(iter, np.around( x0,5)))
    fileptr.writelines("{:.15g}\t{:.15g}\n".format(iter, np.around( x0,5)))
    iter += 1

    while(1):
        if ((np.fabs(x1) <= 1) & (float(np.fabs(x1 - x0)) < LIMIT)):
            break
        elif ((np.fabs(x1) >= 1) & (float(np.fabs(x1 - x0) / np.fabs(x1)) < LIMIT)):
            break

        x2 = aitF(f,x1,x0)
        print("{:.15g}\t{:.15g}\n".format(iter, np.around( x2,5)))
        fileptr.writelines("{:.15g}\t{:.15g}\n".format(iter, np.around( x2,5)))

        x0 = x1
        x1 = x2
        iter += 1
```

```
    fileptr.close()

def steffensen(f,LIMIT):
    fileptr = open("steffensen.txt", "w", encoding='utf-8')
    iter = 0   # 迭代次数
    x0 = input("请输入 steffensen 迭代法的初值：x0 = ")
    x0 = float(x0)
    print("{:.15g}\t{:.15g}\n".format(iter, np.around( x0,5)))
    fileptr.writelines("{:.15g}\t{:.15g}\n".format(iter, np.around( x0,5)))
    iter += 1

    while(1):
        x1 = steF(f,x0)
        print("{:.15g}\t{:.15g}\n".format(iter, np.around( x1,5)))
        fileptr.writelines("{:.15g}\t{:.15g}\n".format(iter, np.around( x1,5)))
        if ((np.fabs(x1) <= 1) & (float(np.fabs(x1 - x0)) < LIMIT)):
            break
        elif ((np.fabs(x1) >= 1) & (float(np.fabs(x1 - x0) / np.fabs(x1)) < LIMIT)):
            break
        x0 = x1
        iter += 1

    fileptr.close()
def main():
    if xlow >= xup:
        print("Error!xlow %g should be smaller than xup %g"%(xlow,xup))
        return

    #ait 迭代法
    # aitken(pF2,LIMIT)

    #ste 迭代法
     # steffensen(pF5,LIMIT)
main()          #执行主程序
```

4 线性方程组的数值解法实验与分析

4.1 Gauss 消去法和列主元消去法比较

4.1.1 第三章题目 2 描述

解以下方程组（计算过程保留 3 位小数）

$$\begin{cases} 2.51x_1 + 1.48x_2 + 4.53x_3 = 0.05 \\ 1.48x_1 + 0.93x_2 - 1.3x_3 = 1.03 \\ 2.68x_1 + 3.04x_2 - 1.48x_3 = -0.53 \end{cases}$$

（1）使用 Gauss 消去法解以上方程组；

（2）使用列主元 Gauss 消去法解以上方程组；

（3）检验（1）和（2）得到的两个解中，哪一个更接近准确解。

（计算过程中保留三位小数，方程的准确解为 $x_1 = 1.4531$ ， $x_2 = -1.589195$ ，$x_3 = -0.2748947$ ）

4.1.2 算法介绍

Gauss 消去法：

算法总共分为两大步骤，第一个步骤是将增广矩阵消元形成上三角矩阵,第二个步骤是从下向上进行回代完成解方程的步骤。

在第一个步骤里总共有三重循环：

第一重循环是找到对角线上的值。

第二重循环是要将当前要消元的主元素变成 1，同时该主元素所在的行同时缩小相应的倍数。

第三、四重循环主元素所在的列，完成消元归零的操作。

回代，从最后一行开始，将剩余的未知数的系数化为 1，这行最后一列的数即为该未知数，将其代入上一行求上一行剩余的未知数，以此往复，直至求完所有解。

列主元 Gauss 消去法：

矩阵进行第 k 次消元前，先进行两个步骤：

一，在第 k1 列对角线及以下的元素中选出绝对值最大者，若其等于零，则 det(A)=0,方程组 Ax=b 不满足 Cramer 法则的条件，停止计算。

二，若最大者不为零，且最大者所在行不是第 k-1 行，则交换第 k1 行与最大者所在行。

然后再用 Gauss 消去法进行消元运算，并回代。

4.1.3 问题分析与求解

Gauss 消去法求解结果：

$$\vec{x} = (1.435,-1.561,-0.274)^T$$

列主元 Gauss 消去法求解结果：

$$\vec{x} = (1.452,-1.587,-0.275)^T$$

检验 Gauss 消去法和列主元 Gauss 消去法：

表 4.1 Gauss 消去法列主元 Gauss 消去法计算结果

	X_1	X_2	X_3
Gauss 消去法	1.435	-1.561	-0.274
列主元 Gauss 消去法	1.452	-1.587	-0.275
准确值	1.453	-1.589	-0.275

4.1.4 结果分析

由数据验证，用列主元 Gauss 消去法得到的解更精确。当系数矩阵上三角的元素与下三角元素的数量级相差很大时，可能出现大数“吃”小数的情况,这时如果采用 Gauss 消去法，把小的数作为主元去消元，除数很小，作除法舍入误差大，会影响后续的消元值，使计算解不可靠。而列主元 Gauss 消去法可以很大程度避免这种情况，计算结果也更接近精确解。

4.1.5 Gauss 消去法代码

```
import math
N = 3   # 解 n 阶方程
DA = [[2.51, 1.48, 4.53, 0.05], [1.48, 0.93, -1.3, 1.03], [2.68, 3.04, -1.48, -0.53]]

def gaussforward(R):   # 向前消元
    r = R
    for l in range(N):
        if r[l][l] == 0:
            print('No Answer')
            return
    for i in range(0, N):
        rii = round(r[i][i], 3)
        for j in range(i, N + 1):
            r[i][j] = round(r[i][j] / rii, 3)   # 行 i 系数除以 ri
```

```
            for k in range(i + 1, N):   # i+1 行以下的处理
                rki = round(r[k][i] / rii, 3)
                for j in range(N + 1):
                    r[k][j] = round(r[k][j] - r[i][j] * rki, 3)   #消去每行第一个非 0 项
        return r

def gaussBackward(r, x):   # 回代
        x[N - 1] = round(r[N - 1][N] / r[N - 1][N - 1], 3)
        for i in range(N - 2, -1, -1):
            sum = 0.0
            for j in range(i + 1, N):
                sum = round(sum + r[i][j] * x[j], 3)   # 各项之和
            x[i] = round((r[i][N] - sum) / r[i][i], 3)   # 计算 xi

x = [0] * N
r = gaussforward(DA)
gaussBackward(r, x)
print("x1=%.4g\t x2=%.4g\t x3=%.4g" % (x[0], x[1], x[2]))
```

4.1.6 Gauss 列主元消去法代码

```
import numpy as np

N = 3   # 解 n 阶方程
DA = np.array([[2.51, 1.48, 4.53, 0.0593], [1.48, 0., -1.3, 1.03], [2.68, 3.04, -1.48, -
0.53]])

def swag(r, k, n):
        ans = np.fabs(r[k][k])
        # 对角线上的值，每行第一个非 0 数
        # 列主元最大行
        max = k

        for i in range(k, n):
            # 排除对角线上为 0 的情况
            if np.fabs(r[i][k]) > ans and np.fabs(r[i][k]) != 0:
                ans = np.fabs(r[i][k])
                max = i

        if max != k:
            # 交换
            for i in range(k, N+1):
```

```
                t = r[k][i]
                r[k][i] = r[max][i]
                r[max][i] = t

def gaussforward(R): #向前消元
    r = R.copy()

    for i in range(N):
        if (r[i][0] == 0):
            print('No Answer')
            return None

        swag(r, i, N) #  找最大列主元并交换
        rii = r[i][i] #  对角线上的值

        for j in range(i, N+1):
            r[i][j] = np.around(r[i][j]/rii, 3) #  行 i 系数除以 ri
        for k in range(i+1, N): # i+1 行以下的处理
            rki = r[k][i]
            for j in range(i, N+1):
                r[k][j] = np.around(r[k][j] - r[i][j]*rki, 3) #  消去每行第一个非 0
项
    return r

def gaussBackward(r, x): #回代
    for i in range(N-1, -1, -1):
        sum = 0.0
        for j in range(i+1, N):
            sum = np.around(sum + r[i][j]*x[j], 3) #  各项之和

        x[i] = np.around((r[i][N] - sum)/r[i][i], 3) #  计算 xi

    return x

x = np.zeros(N)
r = gaussforward(DA)
if r is not None:
    x = gaussBackward(r, x)
    print("x1=%.4g, x2=%.4g, x3=%.4g" % (x[0], x[1], x[2]))
```

4.2 列主元 Gauss-Jordan 消去法、LU 分解法的比较

4.2.1 第三章题目 3 描述

用列主元 Gauss-Jordan 消去法、**LU** 分解法解方程组 $\boldsymbol{Ax}=\boldsymbol{b}$

$$\boldsymbol{A}=\begin{bmatrix}1 & 2 & 1 & -2\\ 2 & 5 & 3 & -2\\ -2 & -2 & 3 & 5\\ 1 & 3 & 2 & 3\end{bmatrix},\quad \boldsymbol{b}=(4,7,-1,0)^{\mathrm{T}}$$

4.2.2 算法介绍

Gauss-Jordan 消去法：

在 Gauss 消去法的基础上，将系数矩阵对角线上的值化为 1，并且消去除对角线上外的所有值。

LU 分解法：

设 **A** 为 n 阶矩阵，若 **A** 的所有顺序主子式均不为 **0**，则 **A** 可分解为一个单位下三角矩阵 **L** 和一个上三角矩阵的乘积，且这种分解是唯一的。

因为

$$A\vec{\mathrm{x}}=LU\vec{\mathrm{x}}=\vec{b_i}(i=1,2,\cdots,k) \tag{4.1}$$

令

$$U\vec{\mathrm{x}}=\vec{y} \tag{4.2}$$

则

$$L\vec{y}=\vec{b_i} \tag{4.3}$$

解出 $\vec{y}$ 代入即得 $\vec{\mathrm{x}}$。

4.2.3 问题求解与分析

Gauss-Jordan 消去法求解结果：

$$\mathrm{x}=(2,-1,2,-1)^{\mathrm{T}}$$

LU 分解法求解结果：

$$\mathrm{x}=(2,-1,2,-1)^{\mathrm{T}}$$

4.2.4Gauss-Jordan 消去法代码

```
import numpy as np
```

```
N = 4
A = np.array([[1, 2, 1, -2, 4], [2, 5, 3, -2, 7], [-2, -2, 3, 5, -1], [1, 3, 2, 3, 0]])

def gaussjordan(A):
    a = A.copy()
    for i in range(N):
        # 找最大列主元并交换
        maxindex = abs(a[i:, i]).argmax() + i
        if a[maxindex, i] == 0:
            print("No unique solution exists")
            return None
        if maxindex != i:
            a[[i, maxindex]] = a[[maxindex, i]]
        # 将主元归一
        a[i] = a[i] / a[i, i]
        # 消元
        for j in range(N):
            if j != i:
                a[j] = a[j] - a[j, i] * a[i]
    return a[:, -1]

x = gaussjordan(A)
if x is not None:
    print("x1 = %.4g, x2 = %.4g, x3 = %.4g, x4 = %.4g" % tuple(x))
```

4.2.5 LU 分解法代码

```
import numpy as np

def LU(A, b):
    n = len(A)
    L = np.zeros(shape=(n, n))
    U = np.zeros(shape=(n, n))

    for base in range(n - 1):
        for i in range(base + 1, n):
            L[i, base] = A[i, base] / A[base, base]
            A[i] = A[i] - L[i, base] * A[base]

    for i in range(n):
        L[i, i] = 1
        U[i] = A[i]
```

```
    print("L:")
    print(L)
    print("U:")
    print(U)

    Y = np.zeros(shape=(n, 1))
    Y[0] = b[0]

    for i in range(n):
        sum3 = 0
        for k in range(i):
            sum3 += L[i][k] * Y[k]
        Y[i] = b[i] - sum3

    X = np.zeros(shape=(n, 1))
    X[n - 1] = Y[n - 1] / U[n - 1][n - 1]

    for i in range(n - 2, -1, -1):
        sum4 = 0
        for k in range(i + 1, n):
            sum4 += U[i][k] * X[k]
        X[i] = (Y[i] - sum4) / U[i][i]

    print("X = ")
    print(X)

A = np.array([[1, 2, 1, -2],
              [2, 5, 3, -2],
              [-2, -2, 3, 5],
              [1, 3, 2, 3]])
b = np.array([[4], [7], [-1], [0]])

LU(A, b)
```

4.3 范式和条件数的求解

4. 3. 1 第三章题目 7 描述

求出

$$\boldsymbol{A}=\begin{bmatrix}-2 & 1 & 0 & 0\\ 1 & -2 & 1 & 0\\ 0 & 1 & -2 & 1\\ 0 & 0 & 1 & -2\end{bmatrix}$$

的$\|\boldsymbol{A}\|_\infty$，$\|\boldsymbol{A}\|_1$，$\|\boldsymbol{A}\|_2$和$\mathrm{Cond}(\boldsymbol{A})_2$。

4.3.2 预备知识

本题调用 numpy 包计算。
np.linalg.norm(求范数)：linalg=linear（线性）+algebra(代数)，norm 则表示范数。
x_norm=np.linalg.norm(x,ord=None, axis=None, keepdims=False)
1. x：表示矩阵（也可以是一维）
2. ord：范数类型
向量的范数：

表 4.2 向量的范数说明

参数	说明	计算方法
ord=2	2-范数	$\sqrt{x_1^{\,2}+x_2^{\,2}+\ldots+x_n^{\,2}}$
ord=1	1-范数	$\lvert x_1\rvert+\lvert x_2\rvert+\ldots+\lvert x_n\rvert$
ord=np.inf	∞-范数	$\max(\lvert x_i\rvert)$

矩阵的范数：

表 4.3 矩阵的范数说明

参数	解释	说明	计算方法
ord=2	求特征值，然后求 max 特征值的算数平方根	2-范数	$\lVert A\rVert_2=\sqrt{\rho(A^TA)}$
ord=1	列和的最大值	1-范数	$\lVert A\rVert_1=\max\limits_{1\le i\le n}\sum\limits_{i=1}^{n}\lvert a_{ij}\rvert$
ord=np.inf	行和的最大值	∞-范数	$\lVert A\rVert_\infty=\max\limits_{1\le i\le n}\sum\limits_{j=1}^{n}\lvert a_{ij}\rvert$

3. axis:处理类型：
   axis = 1 表示按行向量处理，求多个行向量的范数;
   axis = 0 表示按列向量处理，求多个列向量的范数;
   Axis = None 表示矩阵范数。
4. keepding：矩阵的二维特性
True 表示保持知阵的二维特性，False 表示不保持矩阵的二维特性。

4.3.3 问题求解与分析

$\|A\|_\infty = 4.0$

$\|A\|_1 = 4.0$

$\|A\|_2 = 3.6180339887498945$

$\mathrm{Cond}(A)_2 = 9.472135954999578$

结果讨论：

求解方法正确，与 numpy 求解结果一致。

4.3.4 算法源代码

```
import numpy as np
A=[[-2, 1, 0, 0], [1, -2, 1, 0], [0, 1,-2, 1], [0, 0, 1,-2]]
print(np.linalg.norm(A,ord=np.inf))
print(np.linalg.norm(A, ord=1))
print(np.linalg.norm(A, ord=2))
print(np.linalg.cond(A))
```

4.4 微小扰动对方程组求解的稳定性分析

4. 4. 1 第三章题目 8 描述

解如下方程组

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} \boldsymbol{x} = \boldsymbol{b}$$

其中 $\boldsymbol{b} = (32, 23, 33, 31)^{\mathrm{T}}$，但是由于某些原因使得方程的右端被修改为 $\boldsymbol{b}^* = (32.1, 22.9, 33.1, 30.9)^{\mathrm{T}}$，求出方程的解，并算出在∞-范数和 1-范数下求出的解与准确解之间、扰动方程的右端项和原右端项的相对误差，说明原因。

4. 4. 2 预备知识

依题意得，在方程右端被修改后

$$A(\vec{x} + \delta\vec{x}) = \vec{b} + \delta\vec{b} \tag{4.4}$$

进而

$$\delta\vec{x} = A^{-1}\delta\vec{b} \tag{4.5}$$

两边取范数得

$$\delta\vec{x} = \| A^{-1} \| \cdot \| \delta\vec{b} \| \tag{4.6}$$

又

$$\| \vec{b} \| = \| A\vec{x} \| \leq \| A \| \cdot \| \vec{x} \| \tag{4.7}$$

进而

$$\frac{1}{\| \vec{x} \|} \leq \frac{\| A \|}{\| \vec{b} \|} \tag{4.8}$$

所以

$$\frac{\| \delta\vec{x} \|}{\| \vec{x} \|} \leq \| A \| \cdot \| A^{-1} \| \cdot \frac{\| \delta\vec{b} \|}{\| \vec{b} \|} \tag{4.9}$$

以上式子表明解得扰动与右端项的扰动有关系。
本题解方程采用列主元高斯法解方程。

4.4.3 问题求解与分析

未修改的方程解为：

$$x = (1,1,1,1)^{T}$$

修改后的方程解为:

$$x = (9.2,-12.6,4.5,-1.1)^{T}$$

1-范数下

扰动方程的右端项和原右端项相对误差：0.00336134

解与准确解的相对误差：6.85

∞-范数下

扰动方程的右端项和原右端项相对误差：0.0030303

解与准确解的相对误差：13.6

结果讨论：

算法介绍栏已说明解的扰动与右端项扰动的关系，从中可以看出，解的相对误差不超过 b 的相对误差的倍数。右端项的扰动对解的影响与条件数的大小有关，条件数越大，扰动对解的影响越大。

4.4.4 算法源代码

```
import numpy as np
```

```
N = 4

b = [32, 23, 33, 31]
b1 = [32.1, 22.9, 33.1, 30.9]
db = [0] * N
dx = [0] * N

for i in range(N):
    db[i] = b1[i] - b[i]

x1 = [1, 1, 1, 1]
x2 = [9.2, -12.6, 4.5, -1.1]

h1 = np.linalg.norm(db, ord=1)
h11 = np.linalg.norm(b, ord=1)
houl = h1 / h11

h2 = np.linalg.norm(db, ord=np.inf)
h22 = np.linalg.norm(b, ord=np.inf)
hou2 = h2 / h22

for i in range(N):
    dx[i] = x2[i] - x1[i]

q1 = np.linalg.norm(dx, ord=1)
q11 = np.linalg.norm(x1, ord=1)
qian1 = q1 / q11

q2 = np.linalg.norm(dx, ord=np.inf)
q22 = np.linalg.norm(x1, ord=np.inf)
qian2 = q2 / q22

print("右端项和原右端项：1-范数\n")
print("%g\n" % houl)
print("右端项和原右端项：无穷-范数\n")
print("%g\n" % hou2)
print("解：1-范数\n")
print("%g\n" % qian1)
print("解：无穷-范数\n")
print("%g\n" % qian2)
```

4.5 Jacobi 和 Gauss-Seidel 迭代法收敛性

4.5.1 第三章题目 9 描述

设有系数矩阵

（1） $\begin{bmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix}$ （2） $\begin{bmatrix} 2 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & -2 \end{bmatrix}$

分别检验以上系数矩阵用 Jacobi 迭代法和 Gauss-Seidel 迭代法是否收敛。

5.5.2 预备知识

Jacobi 迭代法;

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \tag{4.10}$$

在 a_{ii} 不等于零的情况下化成

$$\begin{cases} x_1 = \dfrac{1}{a_{11}}(-a_{12}x_2 - \cdots - a_{1n}x_n + b_1) \\ x_2 = \dfrac{1}{a_{21}}(-a_{21}x_2 - \cdots - a_{2n}x_n + b_2) \\ \cdots\cdots \\ x_n = \dfrac{1}{a_{nn}}(-a_{n1}x_2 - \cdots - a_{nn-1}x_{n-1} + b_1) \end{cases} \tag{4.11}$$

写成分量形式为

$$x_i^{(k+1)} = \frac{1}{aii}(-\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j) / a_{ii}, i = 1,2,\cdots,n \tag{4.12}$$

写成矩阵形式为

$$\begin{bmatrix} \vec{x}_1^{(k+1)} \\ \vec{x}_2^{(k+1)} \\ \vdots \\ \vec{x}_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & -\frac{a_{13}}{a_{11}} & \cdots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} & \cdots & -\frac{a_{2n}}{a_{22}} \\ -\frac{a_{31}}{a_{33}} & -\frac{a_{32}}{a_{33}} & 0 & \ddots & \vdots \\ \vdots & & \ddots & 0 & -\frac{a_{n-1,n}}{a_{n-1,n-1}} \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \cdots & -\frac{a_{n,n-1}}{a_{nn}} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix} + \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{bmatrix} \tag{4.13}$$

即

$$A\vec{x} = \vec{b} \Leftrightarrow (D + L + U)\vec{x} = \vec{b}$$

$$\Leftrightarrow D\vec{x} = -(L + U)\vec{x} + \vec{b} \quad \text{两边左乘} D^{-1}$$

$$\Leftrightarrow \vec{x} = -D^{-1}(L + U)\vec{x} + D^{-1}\vec{b}$$

得 Jacobi 迭代矩阵

$$x^{(k+1)} = B_J x^{(k)} + \overrightarrow{g_J} \tag{4.14}$$

当 $\vec{x}^{(k+1)}$ 和 $\vec{x}^{k}$ 几乎相同时（差值小于限制范围），得到解。

Gauss-Seidel 迭代法：

此方法是 Jacobi 迭代法的改进，将 *k+1* 次最新计算出来的近似值 $\vec{x}^{(k+1)}$ 的分量 $\vec{x}_n^{(k+1)}$ 加以利用。

用矩阵形式表示如下：

$$(D + L + U)\vec{x} = \vec{b}$$

$$D\vec{x} = -(L + U)\vec{x} + \vec{b}$$

$$D\vec{x}^{(k+1)} = -L\vec{x}^{(k+1)} - U\vec{x}^{(k)} + \vec{b}$$

$$\vec{x}^{(k+1)} = -D^{-1}(L\vec{x}^{(k+1)} + U\vec{x}^{(k)}) + D^{-1}\vec{b}$$

等价于

$$(D + L)\vec{x}^{(k+1)} = -U\vec{x}^{(k)}\vec{b}$$

即

$$\vec{x}^{(k+1)} = -(D+L)^{-1}U\vec{x}^{(k)} + (D+L)^{-1}\vec{b} \tag{4.15}$$

$$\vec{x}^{(k+1)} = B_s + \vec{g}_s \tag{4.16}$$

迭代法收敛基本定理：

设有方程组 $\vec{x} = B\vec{x} + \vec{g}$，对任意初始向量 $\vec{x}^{(0)}$ 几人一常数向量 $\vec{g}$，解此方程组的迭代法（即 $\vec{x}^{(k+1)} = B\vec{x}^{(k)} + \vec{g}, k \to \infty$）收敛的冲要条件是

$$\rho(B) < 1$$

其中 $\rho(B) = \max\limits_{1 \le i \le n} |\lambda_i|$，$\lambda_i$ 为 ***B*** 的特征值。

4.5.3 问题分析与求解

表 4.4 两个矩阵的收敛性验证

使用方法	矩阵(1)的收敛性	矩阵(2)的收敛性
Jacobi 迭代法	不收敛	不收敛
Gauss-Seidel 迭代法	不收敛	收敛

4.5.4 算法源代码

【Jacobi 算法源代码】

```
import numpy as np

N = 3

# 定义 Jacobi Averaging 方法
def jacobiAver(A):
    D = np.zeros((N,N))
    for i in range(N):
        rec = 1.0 / A[i][i]
        D[i][i] = rec
    Ai = np.dot(-D, A)

    # 求特征值，判断是否收敛
    an, _ = np.linalg.eig(Ai)
    an1 = len(an) * [0.0]
    for i in range(len(an)):
        # 取特征值实数和复数的模
```

```
            an1[i] = np.real(abs(an[i]))
        maxx = an1[0]
        for i in range(len(an)):
            # 求特征值的最大值
            maxx = an1[i] if maxx < an1[i] else maxx
        if maxx > -1:
            print("不收敛")
        else:
            print("收敛")

if __name__ == '__main__':
        A = [[1,2,-2],[1,1,1],[2,2,1]]
        B = [[2,-1,1],[1,1,1],[1,1,-2]]
        jacobiAver(A)
        jacobiAver(B)
```

【Gauss-Seidel 算法源代码】

```
import numpy as np

N = 3

def GaussSeidelAver(A):
        D = np.zeros((N, N))
        L = np.zeros((N, N))
        U = np.zeros((N, N))
        for i in range(N):
            # 求 D、L 和 U 矩阵
            D[i][i] = A[i][i]
            for j in range(N):
                if i > j:
                    L[i][j] = -A[i][j]
                elif i < j:
                    U[i][j] = -A[i][j]
        X = D + L
        X1 = np.linalg.inv(X)
        B = np.dot(-X1, U)

        # 求特征值，判断是否收敛
        an, f = np.linalg.eig(B)
        an1 = len(an) * [0.0]
        for i in range(len(an)):
            # 取特征值实数和复数的模
            an1[i] = np.real(abs(an[i]))
```

```
    maxx = an1[0]
    for i in range(1, len(an)):
        # 求特征值的最大值
        maxx = an1[i] if maxx < an1[i] else maxx
    if maxx >= 1:
        print("不收敛")
    else:
        print("收敛")

if __name__ == "__main__":
    #A = [[1, 2, -2], [1, 1, 1], [2, 2, 1]]
    AA = [[2, -1, 1], [1, 1, 1], [1, 1, -2]]
    GaussSeidelAver(AA)
```

4. 6 Jacobi 迭代法和 Gauss-Seidel 迭代法解方程组

4.6.1 第 3 章题目 10 描述

分别用 Jacobi 迭代法和 Gauss-Seidel 迭代法解方程组

$$\begin{cases} 11x_1 - 3x_2 - 2x_3 = 3 \\ -x_1 + 5x_2 - 3x_3 = 6 \\ -2x_1 - 12x_2 + 19x_3 = -7 \end{cases}$$

取初值为 $(0,0,0)^T$，写出前 3 次迭代的结果。

4.6.2 预备知识

Jacobi 迭代法和 Gauss-Seidel 迭代法于本章第五节已介绍，此处不再赘述。

4.6.3 问题分析与求解

本题方程组的准确解为 $\vec{x} = (1,2,1)^T$。

表 4.5 Jacobi 迭代法 3 次迭代的结果

迭代次数	结果
1	$(0.27273,1.2,-0.36842)^T$
2	$(0.53301,1.0335,0.41818)^T$
3	$(0.63062,1.5575,0.34042)^T$

表 4.6 Gauss-Seidel 迭代法 3 次迭代结果

迭代次数	结果
1	$(0.27273,1.2545,0.45263)^T$
2	$(0.69717,1.611,0.72245)^T$
3	$(0.84345,1.8022,0.85857)^T$

结果讨论：

由表 4.5 和表 4.6 的敷据比教得，Guss-Seidel 迭代法比 Jacobi 迭代法更快收敛于准确值。

4.6.4 算法源代码

【Jacobi 迭代法源代码】

```
import numpy as np

N = 3

def jacobi(A, B, x0, x):
    times = 1   # 迭代次数
    f = open("data.txt", "w")   # 打开文件
    while times < 4:   # 要选代 3 次
        for i in range(N):
            k = 0
            for j in range(N):
                if i != j:
                    k += A[i][j] * x[j]   # 算出右端项带有 x 的项的值的相反数
            x[i] = (B[i] - k) / A[i][i]   # 求选代后的 x
            f.write("x{}({}): {:.5g}\t".format(i + 1, times, x[i]))   # 写入文件
        f.write("\n")
        times += 1   # 选代计数
        x0 = x.copy()   # 复制 x 到 x0
    f.close()

if __name__ == "__main__":
    #A = [[1,2,-2],[1,1,1],[2,2,1]]
    A = [[11, 2, -2], [1, 1, 1], [-2, 12, 19]]   # 系数矩阵
    B = [3,6,-7]
    x = [0, 0, 0]   # 初始值
    x0 = x.copy()
    jacobi(A, B, x0, x)
```

【Gauss-Seidel 算法源代码】

```
import numpy as np
```

```
N = 3

def GaussSeidel(A, B, x0):
    times = 1    # 选代次数
    f = open("data.txt", "w")    # 打开文件
    while times <= 3:    # 要选代 3 次
        for i in range(N):
            k = 0
            for j in range(N):
                if i != j:
                    k += x0[j] * A[i][j]    # 算出右端项带有的项的值的相反数
            x0[i] = (B[i] - k) / A[i][i]    # 将新计算的 x 对旧的进行替换
        f.write("({:d})\t{:.5g}\t{:.5g}\t{:.5g}\n".format(times, x0[0], x0[1], x0[2]))
# 写入文件
        times += 1    # 选代计数
    f.close()    # 关闭文件
    return 0

if __name__ == "__main__":
    #A = [[1, 2, -2], [1, 1, 1], [2, 2, 1]]
    A =[[11,-3,-2],[-1,5,-3],[-2,-12,19]] #系数矩阵
    B = [3, 6, -7]
    x0 = [0, 0, 0]    # 初始值
    GaussSeidel(A, B, x0)
```

4. 7 综合实验

4.7.1 实验题目

系数阵为 Hilbert 阵，对解全为 1 的方程组，随着 n = 2, 3, ... 的增加。编写程序，测试和分析利用直接法和迭代法求解方程组的结果差别。

$$H_n = \begin{bmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \cdots & \frac{1}{2n-1} \end{bmatrix}$$

4.7.2 实验准备

1.不同 n 下的方程病态性的分析
2.不同 n 下选代法收敛性的分析
本实验需要的算法都已在上文介绍，此处不再赘述

4.7.3 实验结果与分析

表 4.6 前 20 阶 Hilbert 阵的条数

阶数	1	2	3	4	5
2-条件数	1	19.28	524.1	15510	476600
阶数	6	7	8	9	10
2-条件数	1.4950E+07	4.7540E+08	1.5260E+10	.9320E+11	1.6020E+13
阶数	11	12	13	14	15
2-条件数	5.2230E+14	1.7610E+16	3.1910E+18	9.2760E+17	3.6760E+17
阶数	16	17	18	19	20
2-条件数	7.0630E+17	8.0710E+17	1.4140E+18	5.1900E+18	1.3190E+18

H_2：

表 4.7 二阶 Hilbert 阵各方法下求解结果表

解	直接法			迭代法*		
	Gauss 消去	列主元 Gauss 消去	Gauss-Jordan	LU 分解	Jacobi	Gauss-Seidel
x_1	1	1	1	1	1	1.00002
x_2	1	1	1	1	1	0.999976

H_3：

表 4.8 三阶 Hilbert 阵各方法下求解结果表

解	直接法			迭代法*		
	Gauss 消去	列主元 Gauss 消去	Gauss-Jordan	LU 分解	Jacobi	Gauss-Seidel
x_1	1	1	1	1	不收敛	0.999893
x_2	1	1	1	1	…	1.00055
x_3	1	1	1	1	…	0.999494

H_4：

表 4.9 三阶 Hilbert 阵各方法下求解结果表

解	直接法			迭代法*		
	Gauss 消去	列主元 Gauss 消去	Gauss-Jordan	LU 分解	Jacobi	Gauss-Seidel
x_1	1	1	1	1	不收敛	1.00194
x_2	1	1	1	1	…	0.979153

x_3	1	1	1	1	…	1.0489
x_4	1	1	1	1	…	0.968742

H_5：

表 4.10 五阶 Hilbert 阵各方法下求解结果表

解	直接法			迭代法*		
	Gauss 消去	列主元 Gauss 消去	Gauss-Jordan	LU 分解	Jacobi	Gauss-Seidel
x_1	1	1	1	1	不收敛	1.00079
x_2	1	1	1	1	…	0.995704
x_3	1	1	1	1	…	0.993945
x_4	1	1	1	1	…	1.03277
x_5	1	1	1	1	…	0.975944

H_6：

表 4.11 六阶 Hilbert 阵各方法下求解结果表

解	直接法			迭代法*		
	Gauss 消去	列主元 Gauss 消去	Gauss-Jordan	LU 分解	Jacobi	Gauss-Seidel
x_1	1	1	1	1	不收敛	0.999266
x_2	1	1	1	1	…	1.01302
x_3	1	1	1	1	…	0.953741
x_4	1	1	1	1	…	1.03752
x_5	1	1	1	1	…	1.02957
x_6	1	1	1	1	…	0.966117

H_7：

表 4.12 七阶 Hilbert 阵各方法下求解结果表

解	直接法			迭代法*		
	Gauss 消去	列主元 Gauss 消去	Gauss-Jordan	LU 分解	Jacobi	Gauss-Seidel
x_1	1	1	1	1	不收敛	0.998132
x_2	1	1	1	1	…	1.02522
x_3	1	1	1	1	…	0.932109
x_4	1	1	1	1	…	1.02379
x_5	1	1	1	1	…	1.04851
x_6	1	1	1	1	…	1.01647
x_6	1	1	1	1	…	0.954455

H_{12}：

表 4.13 十二阶 Hilbert 阵各方法下求解结果表

解	直接法			迭代法*		
	Gauss 消去	列主元 Gauss 消去	Gauss-Jordan	LU 分解	Jacobi	Gauss-Seidel

x_1	1	1	1	1	不收敛	0.998236
x_2	1	1	1	1.00001	…	1.0079
x_3	0.999916	0.999916	0.999937	0.999831		1.02045
x_4	1.00112	100085	1.00085	1.00235		0.953532
x_5	0.991859	0993833	0.993833	0.982486		0.968774
x_6	1.03538	1.02684	1.02684	1.07796		1.00455
x_7	0.902315	0.925823	0.925823	0.780349		103083
x_8	1.17542	1.13338	1.13338	1.40142		1.03977
x_9	0.795768	0.844519	0.844519	0.525459		1.03229
x_{10}	1.14866	1.11332	1.11332	1.35009		1.01189
x_{11}	0.938525	0953085	0.953085	0.853503		0.982285
x_{12}	1.01102	1.00842	1.00842	1.02655		0.946618

H_{15}：

表 4.14 十五阶 Hilbert 阵各方法下求解结果表

解	直接法			迭代法*		
	Gauss 消去	列主元 Gauss 消去	Gauss-Jordan	LU 分解	Jacobi	Gauss-Seidel
x_1	1	1	1	1	不收敛	不收敛
x_2	1	0.999995	0.999995	1.00002	…	…
x_3	0.999995	1.00031	1.00031	0.999163		
x_4	0.99864	0.992909	0.992909	1.01446		
x_5	1.02567	1.08108	1.08108	0.865452		
x_6	0.781933	0.461087	0.461087	1.75124		
x_7	2.06684	3.23969	3.23969	-1.64796		
x_8	-2.27904	-5.01454	-5.01454	6.93454		
x_9	7.53239	11.4011	11.4011	-6.94782		
x_{10}	-735505	-9.85871	-9.85871	5.41888		
x_{11}	7.38067	6.0179	6.0179	5.03806		
x_{12}	-1.12904	3.23096	3.23096	-8.97929		
x_{13}	0.425749	-3.5142	-3.5142	9.52431		
x_{14}	1.73328	3.44892	3.44892	-2.59356		
x_{15}	0.817952	0.513499	0.513499	1.6225		

表 4.15 前 20 阶选代矩阵谱半径表

阶数 n	Jacobi 迭代矩阵谱半径	Guass-Seidel 迭代矩阵谱半径
2	0.866025	0.75
3	1.72295	0.980859
4	2.58209	0.99903
5	3.44414	0.999958
6	4.30853	0.999998

7	5.17468	1
8	6.04213	1
9	6.91059	1
10	7.77982	1
11	8.64964	1
12	9.51995	1
13	10.3907	1
14	11.2617	1
15	12.133	1
16	13.0045	1
17	13.8761	1
18	14.748	1
19	15.62	1
20	16.4921	1

*：迭代初值都为 0。

结果讨论:

由表 4.6 可知，前 13 阶中，Hilbet 阵的 2-条件数随着阶数的增加而增加,且随着阶数每增加 1，矩阵的 2-条件数都会增加至少一个数量级。阶数超过 14 时却几乎趋于稳定。当阶数较大时，Hilbert 阵已经接近奇异，计算结果可能不准确。同时，由于 Hilbert 阵条件数呈指数增大趋势。Hilbert 阵是严重病态的。

由表 4.15 可知，Jacobi 迭代矩阵在 n > 2 时谱半径大于 1，迭代不收敛。而 Gauss-Seidel 迭代法谱半径虽表中数据是 1，实际上是趋近于 1。这是因为 Hilbet 阵是对称正定矩阵,根据定理,使用 Gauss-Seidel 迭代法求解一定收敏。

由表 4.6 至表 4.14 可知，低阶的 Hilbert 阵求解，直接法得的解接精确值。这是由于计算时几乎没有舍入误差，所以结果较准确，同时求解速度也很快，当阶数变大(大于 14)时，微小的扰动就会造成极大的误差，无求得解。

迭代法中，Jacobi 造代法除了二阶 Hibet 阵求解都无法收敛，而 Gauss-Seidel 选代法由于停止条件度较低，低阶的 Hibet 阵求解结果不如直接法精确。但当阶数变大(大于 14)时，迭代法依然能够收敛，且解与实际解更为接近，但迭代的次数较多，求解速度较慢。

所以，当要计算低阶 Hilbert 阵求解时，可以采用直接法，结果准确且求解速度快。当要求解阶数较高的 Hilbert 病态方程组时，可以采用 Gauss-Seidel 迭代法，虽然求解速度较慢，但其稳定性保证了求解结果的准确性。

4.7.4 算法源代码

【条件数和谱半径计算源代码】

```
import numpy as np
import math

def Hilbert(N):
    A = np.zeros((N,N))
```

```
    B = np.zeros((N, 1))
    s=0
    for i in range(N):
        for j in range(N):
            A[i][j] = math.pow(i +j+ 1,-1)
            s += A[i][j]
        B[i][0]=s
        s=0
    return A, B

def spectral_radius(M):
    # 求谱半径
    a,b=np.linalg.eig(M) #a 为特征值集合，为特征值向量
    return np.max(np.abs(a)) # 返回谱半径

def jacobiAver(A,N):
    # 确定是否收敛
    D=np.zeros((N,N))
    for i in range(N):
        # 求 D 和(L+U)矩阵
        rec =1.0/A[i][i]
        D[i][i] = rec
        A[i][i] = 0
    B = np.dot(-D,A) # 求此算法的迭代阵
    maxx = spectral_radius(B)
    if maxx >= 1:
        return maxx,0
    else:
        return maxx,1

def GaussSeidelAver(A,N):
    # 确定是否收敛
    D = np.zeros((N,N))
    L = np.zeros((N,N))
    U = np.zeros((N, N))
    for i in range(N):
        # 求 D、L 和 U 矩阵
        D[i][i] = A[i][i]
        for j in range(N):
            if i>j:
                L[i][j]=A[i][j]
            if i<j:
```

```
                U[i][j]=A[i][j]
    X=D+L
    X1 = np.linalg.inv(X)
    B = np.dot(-X1, U)
    maxx = spectral_radius(B)
    if maxx >= 1:
        return maxx,0
    else:
        return maxx,1

f= open('data.txt','w') #  打开文件
for i in range(2, 21):
    A, B = Hilbert(i)
    max1,a = jacobiAver(A,i)
    A1, B = Hilbert(i)
    max2,a = GaussSeidelAver(A1,i)
    f.write("%g\n" % max1)
    f.write("%g\n" % max2)
    # f.write("%.4g\n" % np.linalg.cond(A, 2)) #  条件数数据
    if i%5 == 0:
        f.write('\n')
f.close()
```

【其他算法源代码】

```
import numpy as np
import math

eps = 1e-5


def gauss_forward(R):
    N = len(R)
    r = R.copy()

    for i in range(0, N):
        h = 0
        while abs(r[i][i]) < eps and i < N - 1:
            h += 1
            r[[i, i + h]] = r[[i + h, i]]    # swap rows

        rii = r[i][i]
        for k in range(i + 1, N):
            rki = r[k][i] / rii
```

```
                for j in range(i, N + 1):
                    r[k][j] = r[k][j] - r[i][j] * rki

    if abs(r[N - 1][N - 1]) < eps:
        print('No unique solution')
        return None

    return r


def gauss_backward(r):
    N = len(r)
    x = np.zeros(N)
    x[-1] = r[-1][-1] / r[-1][-2]

    for i in range(N - 2, -1, -1):
        sum = 0.0
        for j in range(i + 1, N):
            sum += r[i][j] * x[j]
        x[i] = (r[i][-1] - sum) / r[i][i]

    return x
# 交换矩阵某两行函数
def swap(r, k, n):
    ans = r[k][k] # 对角线上的值，每行第一个非 0 数
    max = k # 列主元最大行
    for i in range(k, n):
        if ans < np.fabs(r[i][k]):
            ans = np.fabs(r[i][k])
            max = i
    if max != k: # 交换
        for i in range(0, N + 1):
            t = r[k][i]
            r[k][i] = r[max][i]
            r[max][i] = t

def gauss_forward1(R): # 向前消元
    r = R
    for i in range(0, N):
        swap(r, i, N) # 找最大列主元并交换
        rii = r[i][i] # 对角线上的值
        r[i][i] = r[i][i] / rii # 行系数除以 rii
```

```
            for k in range(i+1, N): # +1  行以下的处理
                rki = r[k][i]
                r[k][i] = 0 #  消去每行第一个非 0 项
                for j in range(i+1, N + 1):
                    r[k][j] = r[k][j] - r[i][j] * rki / rii
        return r

def gauss_backward1(r, x): #  回代
        for i in range(N - 1, -1, -1):
            sum = 0.0
            for j in range(i + 1, N):
                sum = sum + r[i][j] * x[j] #  各项之和
            x[i] = (r[i][N] - sum) / r[i][i] #  计算 xi
        return x

def gauss_jordan(R):
        r = R
        a = [0] * N
        for i in range(0, N):
            if np.fabs(r[i][i]) < eps:
                print('No Answer')
                return 0
            swap(r, i, N)
            rii = r[i][i] #  对角线上的值
            for j in range(i, N+1):
                r[i][j] = r[i][j] / rii #  行 i 系数除以 rii
            for k in range(0, N): # i+1  行以下的处理
                if k != i:
                    rki = r[k][i]
                    for j in range(i, N + 1):
                        r[k][j] = r[k][j] - r[i][j] * rki
            for h in range(0, N):
                a[h] = r[h][N]
        return a
def LU(A, b):
        n = len(A)
        L = np.zeros(shape=(n, n))
        U = np.zeros(shape=(n, n))
        for base in range(n - 1):
            for i in range(base + 1, n):
                L[i, base] = A[i, base] / A[base, base]
                A[i] = A[i] - L[i, base] * A[base]
```

```
    for i in range(n):
        L[i, i] = 1
        U = np.array(A)
    Y = np.zeros(shape=(n, 1))
    Y[0] = b[0]
    for i in range(n):
        sum3 = 0
        for k in range(i):
            sum3 = sum3 + L[i][k] * Y[k]
        Y[i] = b[i] - sum3
    X = np.zeros(shape=(n, 1))
    X[n-1] = Y[n-1] / U[n-1][n-1]
    for i in range(n-2, -1, -1):
        sum4 = 0
        for k in range((i + 1), n):
            sum4 = sum4 + U[i][k] * X[k]
        X[i] = (Y[i] - sum4) / U[i][i]
    return X
def LU(A, b):
    n = len(A)
    L = np.zeros(shape=(n, n))
    U = np.zeros(shape=(n, n))
    for base in range(n - 1):
        for i in range(base + 1, n):
            L[i, base] = A[i, base] / A[base, base]
            A[i] = A[i] - L[i, base] * A[base]
    for i in range(n):
        L[i, i] = 1
        U = np.array(A)
    Y = np.zeros(shape=(n, 1))
    Y[0] = b[0]
    for i in range(n):
        sum3 = 0
        for k in range(i):
            sum3 = sum3 + L[i][k] * Y[k]
        Y[i] = b[i] - sum3
    X = np.zeros(shape=(n, 1))
    X[n-1] = Y[n-1] / U[n-1][n-1]
    for i in range(n-2, -1, -1):
        sum4 = 0
        for k in range((i + 1), n):
            sum4 = sum4 + U[i][k] * X[k]
```

```
            X[i] = (Y[i] - sum4) / U[i][i]
        return X
def GaussSeidel(A, B, x0):
        N = len(A)
        eps = 1e-10 #  收敛判据
        times = 1 #  选代次数
        while times < 1000: #  送代次数
            temp = x0.copy() #  复制旧的 x 到 temp
            for i in range(N):
                k = 0.0
                for j in range(N):
                    k += x0[j][0] * A[i][j] #  算出右端项带有 x 的项的值的相反数
                x0[i][0] = (B[i][0] - k) / A[i][i] #  将新计算的值对旧的进行替换
            norm = np.linalg.norm(x0 - temp) #  计算误差
            times += 1 #  迭代计数
            if norm < eps: #  如果误差小于收敛判据，停止迭代
                break
        if times < 1000: #  如果迭代次数小于 1000，说明迭代收敛
            return x0
        else: #  否则，迭代失败
            return None
def jacobiAver(A):
        #  确定是否收敛
        D = np.zeros((N,N))
        for i in range(N):
            #  求 D 和(L+U)矩阵
            rec = 1.0 / A[i][i]
            D[i][i] = rec
            for j in range(N):
                if i != j:
                    D[i][j] = 0.0
        B = np.dot(-D,A)   #  求此算法的迭代矩阵
        an, ff = np.linalg.eig(B)   #  求特征值
        an1 = len(an)* [0.0]
        for i in range(len(an)):   #  取特征值实数和复数的模
            an1[i] = np.real(abs(an[i]))
        maxx = an1[0]
        for i in range(1, len(an)):   #  求特征值的最大值
            if maxx < an1[i]:
                maxx = an1[i]
        if maxx >= 1:
            return 0
```

```
    else:
        return 1
def GaussSeidelAver(A):
    D = np.zeros((N,N))
    L = np.zeros((N,N))
    U = np.zeros((N,N))

    for i in range(N):
        #求 D、L 和 U 矩阵
        D[i][i] = A[i][i]
        for j in range(N):
            if j < i:
                L[i][j] = A[i][j]
            elif j > i:
                U[i][j] = A[i][j]

    X = D + L
    X1 = np.linalg.inv(X)
    B = np.dot(-X1, U)
    an, f = np.linalg.eig(B) #  求特征值
    an1 = len(an) * [0.0]

    for i in range(len(an)): #  取特征值实数和复数的模
        an1[i] = np.real(abs(an[i]))

    maxx = an1[0]
    for i in range(1,len(an)): #  求特征值的最大值
        if maxx < an1[i]:
            maxx = an1[i]

    if maxx >= 1:
        return 0
    else:
        return 1
if __name__ == '__main__':
    A = np.zeros((N,N))
    B = np.zeros((N,1))
    for i in range(N):
        for j in range(N):
            A[i][j] = math.pow(i + j + 1, -1)
        B[i][0] = sum(A[i])
```

```
da = np.hstack([A, B])
AA = A.copy()
BB = B.copy()

# Gaussian elimination
x = [0]*N
r = gaussforward(da)
if type(r) != int:
	gaussBackward(r,x)
for i in range(N):
	print("x%d = %g" % (i, x[i]))

# Gaussian elimination with partial pivoting
DAA = da.copy()
x = [0]*N
r = gaussforward1(DAA)
if type(r) != int:
	gaussBackward1(r,x)
for i in range(N):
	print("x%d = %g" % (i, x[i]))

# Gauss-Jordan elimination
D = da.copy()
gj = gaussjordan(D)
if type(gj) != int:
	for i in range(N):
		print("x%d = %g" % (i, gj[i][N]))

# LU decomposition
lu = LU(AA,BB)
print("LU decomposition:")
for i in range(N):
	print("x%d = %g" % (i, lu[i]))

AA1 = A.copy()
BB1 = B.copy()
x0 = np.zeros((N,1))
xx = np.zeros((N,1))

# Jacobi iterative method
if jacobiAver(AA1):
	AA2 = A.copy()
```

```
        jacobi(AA2, BB1, x0, xx)
        for i in range(N):
            print("x%d = %g" % (i, xx[i][0]))
    else:
        print("Matrix does not converge! Cannot iterate.")

    AA4 = A.copy()

    # Gauss-Seidel iterative method
    if GaussSeidelAver(AA4):
        AA3 = A.copy()
        BB2 = B.copy()
        x00 = np.zeros((N,1))
        GaussSeidel(AA3, BB2, x00)
        for i in range(N):
            print("x%d = %g" % (i, x00[i][0]))
    else:
        print("Matrix does not converge! Cannot iterate.")
```

5 插值法的实验与分析

5.1 计算题

5.1.1 题目 1

利用拉格朗日插值法，取节点 $x_0=2$， $x_1=2.5$， $x_2=4$，对函数 $f(x)=\frac{1}{x}$ 建立二次插值多项式，计算 $f(3)$ 的近似值并估计误差。

解：由题目知 $f(x)=\frac{1}{x}$，则 $f^{'''}(x)=-\frac{6}{x^4}$。利用题目给的三个点可建立三个插基函数分别为

$$l_0(x)=\frac{(x-2.5)(x-4)}{(2-2.5)(2-4)}=x^2-1.5x+10$$

$$l_1(x)=\frac{(x-2)(x-4)}{(2.5-2)(2.5-4)}=-\frac{4}{3}x^2+8x-\frac{32}{3}$$

$$l_2(x)=\frac{(x-2)(x-2.5)}{(4-2)(4-2.5)}=\frac{1}{3}x^2-\frac{3}{2}+\frac{5}{3}$$

再以 y_0， y_1， y_2 作为组合系数可得插值函数

$$L(x)=\frac{\left[(x-2.5)(x-4)\right]}{2}-\frac{2\cdot2\cdot2\left[(x-2)(x-4)\right]}{5\cdot3}+\frac{\left[(x-2)(x-2.5)\right]}{4\cdot3}$$

$$=\frac{1}{20}x^2-\frac{17}{40}x+\frac{207}{180}$$

带入 x=3 可得 $L(x)\approx0.325$，即 $f(3)\approx0.325$

【误差计算】
已知

$$R(x)=f(x)-\varphi(x)=\frac{f'''(\xi)}{3!}(x-x_0)(x-x_1)(x-x_2)$$

解得差值函数为

$$R(3)=\frac{1}{2\xi^4}$$

$\xi\in(2,4)$ 时， $\frac{1}{512}<R(3)<\frac{1}{32}$

5.1.2 题目 2

已知下列两组插值点

a) $A(0,1)$， $B(1,2)$， $C(2,3)$；

b) $A(1,0)$， $B(3,2)$， $C(4,15)$， $D(7,12)$；

（1）利用拉格朗日插值法分别求通过这些插值点的插值多项式；
（2）构造差商表，利用牛顿法求通过这些插值点的插值多项式。

在上述结果的基础上，如果再增加一点 $(4,4)$，那么应该采用那种方法建立插值多项式？为什么？
解：(1)利用题目给的三个点可建立三个插基函数分别为

$$l_0(x)=\frac{(x-1)(x-2)}{(0-1)(0-2)}$$

$$l_1(x)=\frac{(x-0)(x-2)}{(1-0)(1-2)}$$

$$l_2(x)=\frac{(x-0)(x-1)}{(2-0)(2-1)}$$

再以 y_0， y_1， y_2 作为组合系数可得插值函数

$$L_1(x)=1\cdot\frac{(x-1)(x-2)}{(0-1)(0-2)}+2\cdot\frac{(x-0)(x-2)}{(1-0)(1-2)}+3\cdot\frac{(x-0)(x-1)}{(2-0)(2-1)}$$

化简得 $L_1(x)=x+1$。

同理可得

$$l_0(x)=\frac{(x-4)(x-3)(x-7)}{(1-4)(1-3)(1-7)}$$

$$l_1(x)=\frac{(x-4)(x-1)(x-7)}{(3-4)(3-1)(3-7)}$$

$$l_2(x)=\frac{(x-1)(x-3)(x-7)}{(4-1)(4-3)(4-7)}$$

$$l_3(x)=\frac{(x-1)(x-3)(x-4)}{(7-1)(7-3)(7-4)}$$

$$L(x)=0+2\cdot\frac{(x-4)(x-1)(x-7)}{(3-4)(3-1)(3-7)}+15\cdot\frac{(x-1)(x-3)(x-7)}{(4-1)(4-3)(4-7)}+12\cdot\frac{(x-1)(x-3)(x-4)}{(7-1)(7-3)(7-4)}$$

化简得 $L_2(x)=-1.25x^3+14x^2-38.75x+26$。

(2)

x	$f(x)$	一阶差商	二阶差商
0	1	$f[0,1]=\frac{1-2}{0-1}=1$	$f[0,1,2]=\frac{f[0,1]-f[1,2]}{0-2}=0$
1	2		
2	3	$f[1,2]=\frac{2-3}{1-2}=1$	

因此 $N(x)=1+f[0,1](x-0)+f[0,1,2](x-0)(x-1)=x+1$。

如果再增加一个插值点，应该采用牛顿法构造插值多项式。因为牛顿法在构造差商表后，新增加一个插值点只需要计算一个差商，而拉格朗日插值法则需要重新计算所有的基函数，计算量更大。

5.1.3 题目 3

已知函数 $f(x)$ 有函数表：

x_i	0	1	2	3
$f(x_i)$	0	0	0	0

求满足下列条件 $S''(0)=1$， $S''(3)=0$ 的三次样条插值函数 $S(x)$：

欧炜祺、

解：由题意知，

该函数可分为三个子区间 $[0,1]$，$[1,2]$，$[2,3]$。$(n=3)$。

$S(0)=0$，$h_0=x_1-x_0=1$，$d_0=0$。

$S(1)=S(2)=S(3)=0$，$h_1=h_2=h_3=1$，$d_1=d_2=d_3=0$。

$u_1=u_2=u_3=0$。

根据自然边界条件可知。

$m_0=S''(x_0)=1$，$m_3=S''(x_3)=0$。

代入式子 $u_i=h_{i-1}m_{i-1}+2(h_{i-1}+h_i)m_i+h_im_i$。

可得 $\begin{cases}4m_1+m_2=-1\\ m_1+4m_2=0\end{cases}\Rightarrow\begin{cases}m_1=-\frac{4}{15}\\ m_2=\frac{1}{15}\end{cases}$。

代入 $S_{i,1}=d_i-\frac{h_i(2m_i+m_{i+1})}{6}$，$S_{i,2}=\frac{m_i}{2}$，$S_{i,3}=\frac{m_{i+1}-m_i}{6h_i}$

$\Rightarrow$ $S_{0,1}=-\frac{13}{45}$，$S_{0,2}=\frac{1}{2}$，$S_{0,3}=\frac{-17}{90}$

$S_{1,1}=\frac{7}{90}$，$S_{1,2}=\frac{-2}{15}$，$S_{1,3}=\frac{1}{18}$

$S_{2,1}=\frac{-2}{90}$，$S_{2,2}=\frac{1}{30}$，$S_{2,3}=\frac{-1}{90}$

即 $S_0(x)=-\frac{13}{45}x+\frac{1}{2}x^2-\frac{19}{90}x^3$。

$S_1(x)=\frac{7}{90}(x-1)-\frac{2}{15}(x-1)^2+\frac{1}{18}(x-1)^3$。

$S_2(x)=-\frac{2}{90}(x-2)+\frac{1}{30}(x-2)^2-\frac{1}{90}(x-3)^3$

经检验，$S_0(0)=f(0)$，$S_0(1)=S_1(1)=f(1)$。$S_1(2)=S_2(2)=f(2)$。

$S_0'(1)=S_1'(1)$，$S_0''(1)=S_1''(1)$ $S_1'(2)=S_2'(2)$，$S_1''(2)=S_2''(2)$

满足要求。

5.2 综合实验：物体运动轨迹的插值预测

5.2.1 实验题目

在实际应用中，一般不能确知物体的运动轨迹方程，只能通过测量一些 x 坐

标的值，通过数据插值的方法来求其他未知点的函数值。

采用拉格朗日插值、分段线性插值、样条插值等方法进行插值，估计 x = -3.75 和 0.25 位置的 y 坐标值，并绘制插值函数的图形，根据结果，确定一种最好的插值方法。

实验数据如下表

表 5.1 不同 x 点处物体的位置

x	-5.0	-4.5	-4.0	-3.5	-3.0	-2.5	-2.0	-1.5	-1.0	-0.5
物体 1(y1)	-0.1923	-0.2118	-0.2353	-0.2642	-0.3	-0.3448	-0.4000	-0.4615	-0.5000	-0.4000
物体 2(y2)	0.0016	0.002	0.0025	0.0033	0.0044	0.0064	0.0099	0.0175	0.0385	0.1379

x	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
y1	0	0.4000	0.5000	0.4615	0.4000	0.3448	0.3000	0.2542	0.2353	0.2118	0.1923
y2	1.0000	0.1379	0.0385	0.0175	0.0099	0.0064	0.0044	0.0033	0.0025	0.0020	0.0016

5.2.2 算法介绍

【拉格朗日插值法】

一般地，若已知 $y = f(x)$在互不相同 n+1 个点 $x_0,x1,\ldots,x_n$ 处的函数值 $y_0,y_1,\ldots,y_n$(即该函数过$(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$这 n+1 个点)，则可以考虑构造一个过这 n+1 个点的、次数不超过 n 的多项式 $y= P_n(x)$使其满足:

$$P_n(x_k)=y_k，\ k=0,1,\ldots,n \tag{5.1}$$

要估计任一点ξ，$\xi \neq x_i, i=0,1,...,n$，则可以用 $P_n(\xi)$ 的值作为准确值 $f(\xi)$ 的近似值，此方法叫做“插值法”。

称式(6.1)为插值条件（准则），含 $x_0,x1,\ldots,x_n$ 的最小区间[a,b]，其中 $a = min\{x_0,x1,\ldots,x_n\}$, $b = max\{x_0,x1,\ldots,x_n\}$。

在平面上有$(x_0,y_0),(x_1,y_1),\ldots,(x_{n-1},y_{n-1})$共 n 个点，现做一函数 $f(x)$使其图像经过这 n 个点。

做法：设集合 D_n 是关于点(x,y)的角标的集合，$D_n = \{0,1,\ldots,n\}$，作 n 各多项式 $p_j(x), j \in D_n$。对于任意 $k \in D_n$，都有 $p_k(x)$，$B_k = \{i \mid i \neq k, i \in D_k\}$ 。

使得

$$p_k(x) = \prod_{i \in B_k} \frac{x - x_i}{x_k - x_i} \tag{5.2}$$

$p_k(x)$是 n-1 次多项式，且满足 $\forall m \in B_k$，$p_k(x_k) = 1$，

最后可得

$$L_n(x) = \sum_{j=0}^{n-1} y_i p_i(x) \tag{5.3}$$

形如上式的插值多项式 $L_n(x)$称为拉格朗日(Lagrange)插值多项式。

【牛顿插值法】

函数 $f(x)$的差商定义为

$$f[x_k] = f(x_k)$$

$$f[x_{k-1}, x_k] = \frac{f[x_{k-1}, x_k]}{x_k - x_{k-1}} \tag{5.4}$$

构造高次差商递推公式

$$f[x_j, x_{k-j+1}, ..., x_k] = \frac{f[x_{k-j+1}, ..., x_k] - f[x_{k-j}, ..., x_{k-1}]}{x_k - x_{k-j}} \tag{5.5}$$

设 $x_0,x1,\ldots,x_N$ 是区间[a,b]内 N+1 个不同的数，存在唯一的至多 N 次牛顿多项式

$$P_N(x) = a_0 + a_1(x - x_0) + ... + a_N(x - x_0)(x - x_1)...(x - x_{N-1}) \tag{5.6}$$

其中 $a_k = f[x_0,x1,\ldots,x_k], k=0,1,\ldots,N$。

分段线性插值：

给定 n+1 个插值节点 $x_0<x1<\ldots<x_n$ 以及在这些值节点上的函数值 $y_0,y_1,\ldots,y_n$，构造一个分段插值函数 P(x),使得对任意 i=0,1,2,…,n，有 $P(x_i) = y_i$，并且在每个子区间$[x_i,x_{i+1}]$内，P(x)都是线性函数。称为对插值点 $(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$的分段线性插值函数。

【3 次样条插值算法】

1. 计算点与点之间的步长:

$$h_i = x_{i+1} - x_i (i = 0,1,...,n+1) \tag{5.7}$$

2. 将路径点和端点条件（如果是自由边界三次样条中端点条件即 S 的二阶导为 0）代入如下矩阵方程中:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & 0 & & \\ 0 & h_1 & 2(h_1+h_0) & h_2 & & \\ 0 & 0 & \ddots & \ddots & & \vdots \\ \vdots & & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & & \cdots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_n \end{bmatrix} = 6 \begin{bmatrix} 0 \\ \frac{y_2 - y_1}{h} - \frac{y_1 - y_0}{h} \\ \frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\ \vdots \\ \frac{y_n - y_{n-1}}{h_{n-1}} - \frac{y_{n-1} - y_{n-2}}{h_{n-2}} \\ 0 \end{bmatrix} \tag{5.8}$$

3. 解矩阵方程，求得二次微分值 m_i。
4. 计算每一段的三次样条曲线系数:

$$a_i = y_i$$

$$b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{2} m_i - \frac{h_i}{6}(m_{i+1} - m_i)$$

$$c_i = \frac{m_i}{2}$$

$$d_i = \frac{m_{i+1} - m_i}{6h_i} \tag{5.9}$$

5. 那么在每一个子区间 $x_i \le x \le x_{i+1}$ 内，其对应的样条函数表达式为:

$$f_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{5.10}$$

5.2.3 实验结果及分析

表 5.2 物 1 拉格朗日插值法和插值法下的预测值

插值多项式次数	拉格朗日法		牛顿法	
	-3.75	0.25	-3.75	0.25
1	-0.24975	0.2	-0.24975	0.2
2	-0.2488875	0.2375	-0.2488875	0.2375
3	-0.2489813	0.21875	-0.24898125	0.21875
4	-0.2489953	0.22956641	-0.24899531	0.22956641
5	-0.2490106	0.2241582	-0.24901055	0.2241582
6	-0.2489866	0.22888965	-0.24898662	0.22888965
20	-0.3856162	0.230035	-0.38561617	0.2300346

表 5.3 物体 2 拉格朗日插值法和牛顿插值法下的预测值

插值多项式次数	拉格朗日法预测观测值		牛顿法预测观测值	
	-3.75	0.25	-3.75	0.25
1	0.0004	0.56895	0.0004	0.56895
2	0.0009875	0.4736125	0.0009875	0.4736125
3	0.0028625	0.62904375	0.0028625	0.62904375
4	0.002876563	0.55471875	0.002876563	0.55471875
5	0.002883594	0.65016797	0.002883594	0.65016797
6	0.002901367	0.59188916	0.002901367	0.59188916
20	-12.647852	0.704396	-12.647852	0.70439579

表 5.4 分段线性插值、分段多项式插值和 3 次样条插值的预测值

物体	插值类型	预测-3.75	预测 0.25
1	线性插值	-0.24975	0.2
	二次多项式插值	-0.2489861	0.22375754

	三次多项式插值	-0.24899129	0.22658909
	3 次样条插值	-0.24899129	0.22658909
2	线性插值	0.0029	0.56895
	二次多项式插值	0.00286478	0.64972932
	三次多项式插值	0.002837041	0.66250815
	3 次样条插值	0.002837041	0.66250815

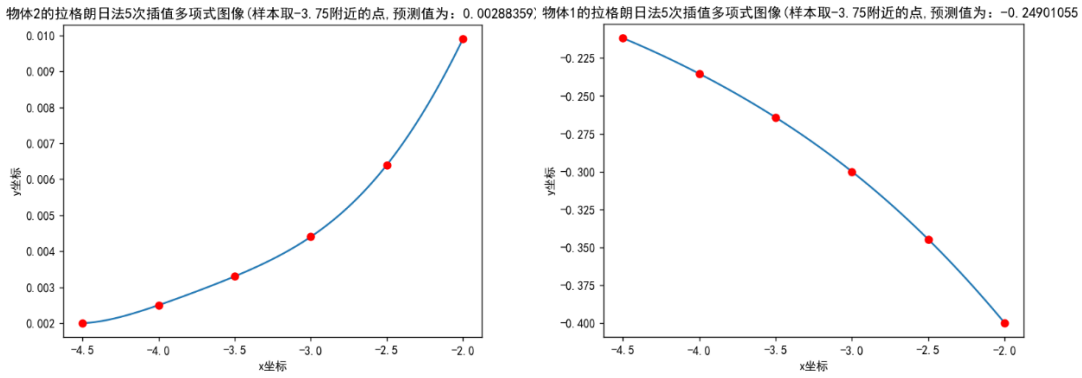


图 5.1　物体 1 和物体 2 五次拉格朗日插值多项式图像（样本点于-3.75 附近）

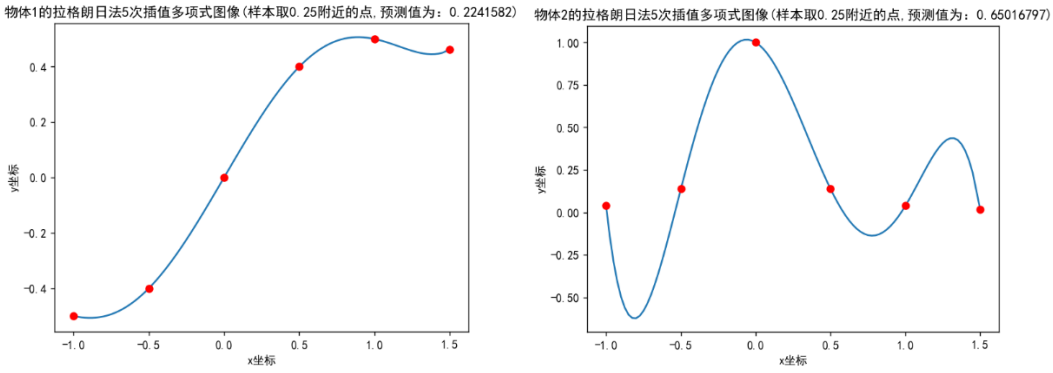


图 5.2　物体 1 和物体 2 五次拉格朗日插值多项式图像（样本点于 0.25 附近）

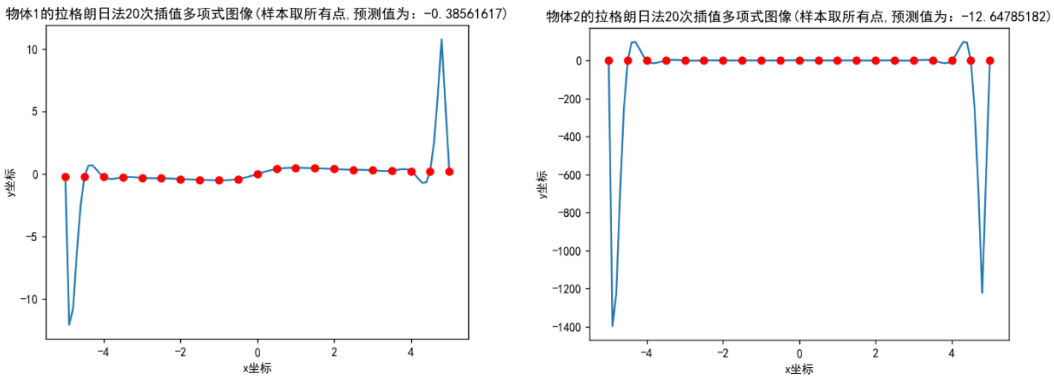


图 5.3 物体 1、2 二十次拉格朗日插值多项式图像（样本点取全部数据点）

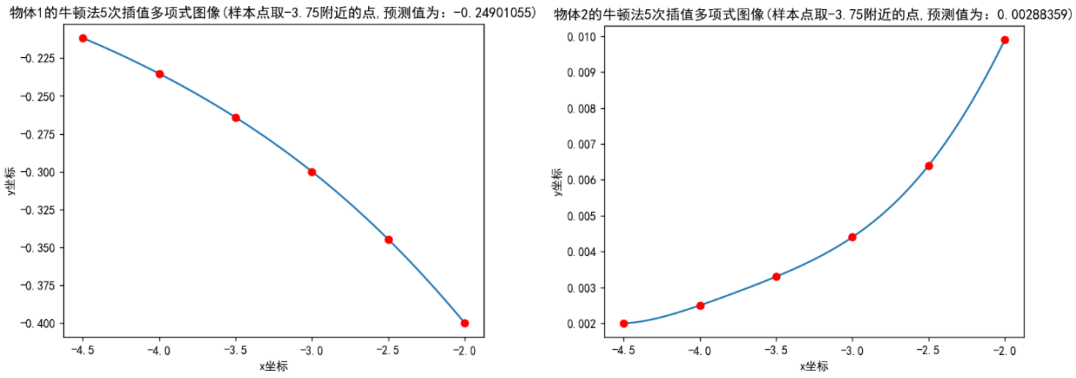


图 5.4　物体 1 和物体 2 五次牛顿插值多项式图像（样本点于-3.75 附近）

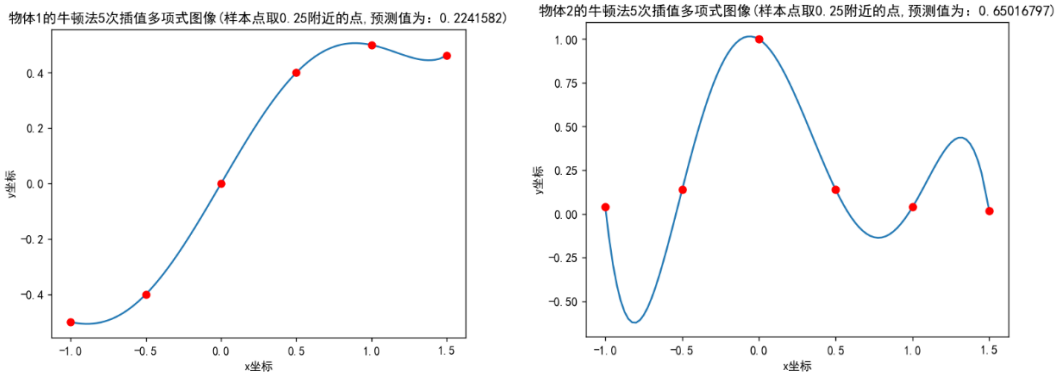


图 5.5　物体 1 和物体 2 五次牛顿插值多项式图像（样本点于 0.25 附近）

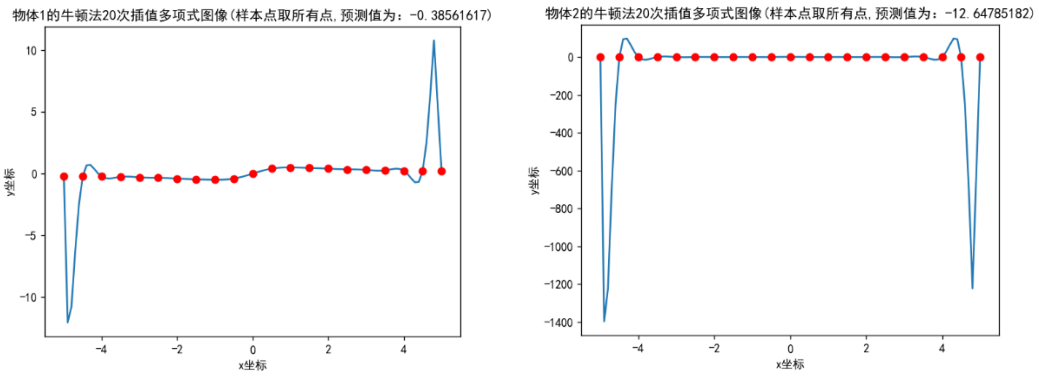


图 5.6　物体 1 和物体 2 二十次牛顿插值多项式图像（样本点取全部数据点）

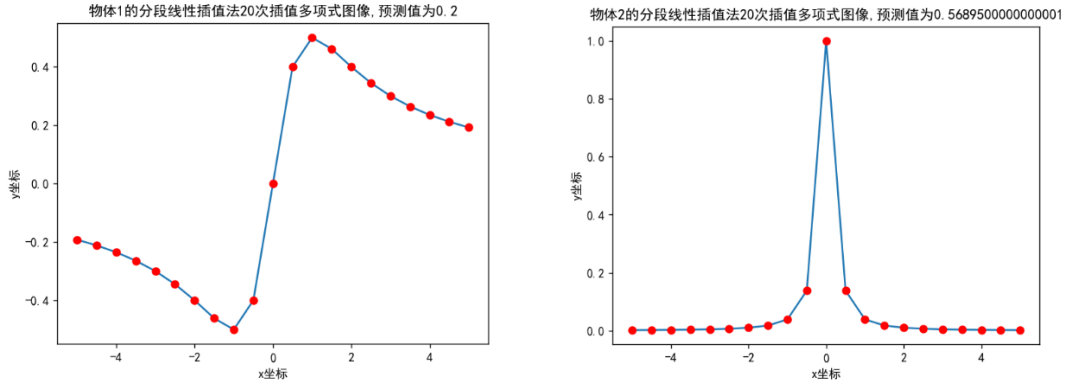


图 5.7　物体 1、2 二十次分段线性插值多项式图像（样本点取全部数据点）

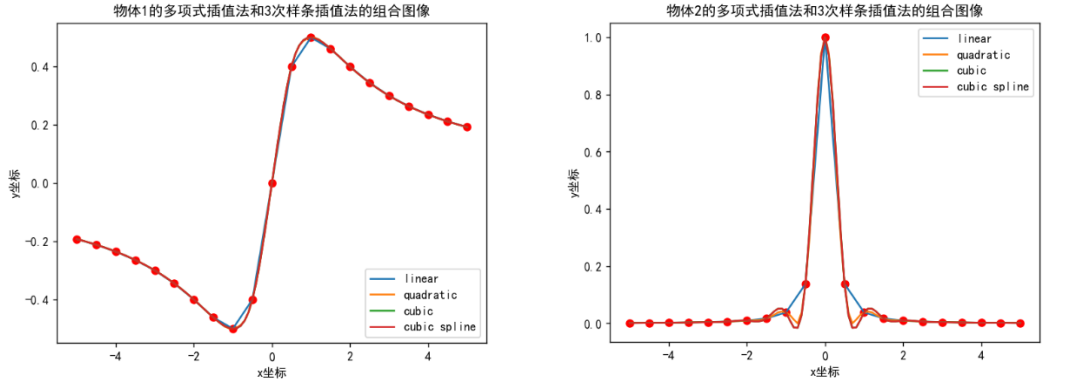


图 5.8　物体 1 和物体 2 多项式插值法和 3 次样条插值法的组合图像

【结果讨论】

将图 5.1 至图 5.8 与题干数据来源的函数图像作对比，不难发现，3 次样条插值和分段三次多项式插值图像与准确结果图像最为接近，故现以 3 次样条插值法下计算得到的预测值作为本次实验结果，即物体 1 在-3.75 和 0.25 的预测值分别为-0.24899129，0.22658909；即物体 2 在-3.75 和 0.25 的预测值分别为 0.002837041，0.66250815。

由图 5.3 和图 5.6 观察得，当拉格朗日插值法和牛顿插值法使用高次多项式插值时，出现了龙格现象。导致数据计算量大，稳定性差，且区间端点处误差很大，预测效果不不理想。

由表 5.2 和表 5.3 得，拉格朗日插值法和牛顿插值法在精确度为 10-5 时结果一致，这也一定程度上验证了两者作为多项式插值法，从本质上说，两者给出的结果是一样的（相同的次数，相同的系数多项式)，只不过表示的形式不同。

由表 5.2 和表 5.3 得，各次插值多项式下拉格朗日插值法和牛顿插值法得到的预测值与准确值相比精度只能达到 10^{-3}，在对精度有高要求时，应优先考虑其他方法。

由图 5.7 可以看到，分段线性插值法的插值图像与实际数据图像形状相近。效果虽好，但因为导数不连续，导致函数光滑性较差。

由图 5.8 可以看到，分段多次多项式样条插值与准确结果图像非常接近，特别是

3 次样条插值和分段三次多项式插值图像与准确结果图像几乎一致，计算得到的预测值的精度也是各个方法中最高的。由此得出，本次数据预测采用 3 次样条插值法或分段三次多项式插值法最为理想。

5.1.4 附录：源代码

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import numpy as np
from scipy import interpolate
import pylab as pl
from sklearn.metrics import mean_squared_error
import matplotlib

matplotlib.rcParams['font.sans-serif'] = ["SimHei"]
matplotlib.rcParams["axes.unicode_minus"] = False


# 物体一的运动方程
def target1(t):
    return t / (1 + t ** 2)


# 物体二的运动方程
def target2(t):
    return 1 / (1 + 25 * t ** 2)


# 完整点
x = [-5.0, -4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5,
4.0, 4.5, 5.0]
# 物体一的完整点
y1 = [-0.1923, -0.2118, -0.2353, -0.2642, -0.3, -0.3448, -0.4000, -0.4615, -0.5000, -0.400,
0, 0.4, 0.5, 0.4615, 0.4,
      0.3448, 0.3, 0.2642, 0.2353, 0.2118, 0.1923]
# 物体二的完整点
y2 = [0.0016, 0.002, 0.0025, 0.0033, 0.0044, 0.0064, 0.0099, 0.0175, 0.0385, 0.1379, 1.0,
0.1379, 0.0385, 0.0175,
      0.0099, 0.0064, 0.0044, 0.0033, 0.0025, 0.002, 0.0016]

# 分成 10，12，14 个点进行测试
x_1 = [-4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0, 0.5]
```

```
x_2 = [-4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0, 0.5, 1.0]
x_3 = [-5.0, -4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0, 0.5, 1.0, 1.5]

# 物体一对应的运动轨迹
y_1 = [-0.2353, -0.2642, -0.3, -0.3448, -0.4000, -0.4615, -0.5000, -0.400, 0, 0.4]
y_2 = [-0.2118, -0.2353, -0.2642, -0.3, -0.3448, -0.4000, -0.4615, -0.5000, -0.400, 0, 0.4, 0.5]
y_3 = [-0.1923, -0.2118, -0.2353, -0.2642, -0.3, -0.3448, -0.4000, -0.4615, -0.5000, -0.400, 0, 0.4, 0.5, 0.4615]

# 物体二对应的运动轨迹
z_1 = [0.0025, 0.0033, 0.0044, 0.0064, 0.0099, 0.0175, 0.0385, 0.1379, 1.0, 0.1379]
z_2 = [0.002, 0.0025, 0.0033, 0.0044, 0.0064, 0.0099, 0.0175, 0.0385, 0.1379, 1.0, 0.1379, 0.0385]
z_3 = [0.0016, 0.002, 0.0025, 0.0033, 0.0044, 0.0064, 0.0099, 0.0175, 0.0385, 0.1379, 1.0, 0.1379, 0.0385, 0.0175]

# 线性插值法，物体 1，所有点进行插值（只是对两端点进行连接预测，因此使用不同点结果相同）
# x_u = x
# y = y1
# target = target1
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# 用于估计预测值和准确值之间的误差
# tag_y = target(xnew)
# 画出当前的曲线
# pl.plot(x_u, y, "ro")
# pl.title('线性插值法')
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f = interpolate.interp1d(x_u, y, kind="linear")
# # 选择对应方式的分段插值
# # 把 x 值代入插值函数，得到 y 坐标用于画出插值曲线
# ynew = f(xnew)
# -3.75 位置的点
# pl.scatter(-3.75, f(-3.75), color='m')
# pl.text(-3.75, f(-3.75), 'interpolate')
# 0.25 位置的点
# pl.scatter(0.25, f(0.25), color='g')
# pl.text(0.25, f(0.25), 'interpolate')
# print("{} error:{}".format("linear", mean_squared_error(tag_y, ynew)))
```

```
# pl.plot(xnew, ynew, label=str("linear"))  # Label 用来显示图例
# # #  目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  #  显示图例的位置
# pl.show()

#  线性插值法，物体 1，所有点进行插值（只是对两端点进行连接预测，因此使用不同点结果相同）
# x_u = x
# y = y2
# target = target2
# #  用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# #  用于估计预测值和准确值之间的误差
# tag_y = target(xnew)
# #  画出当前的曲线
# pl.plot(x_u, y, "ro")
# pl.title('线性插值法')
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f = interpolate.interp1d(x_u, y, kind="linear")
# #  选择对应方式的分段插值
# #  把 x 值代入插值函数，得到 y 坐标用于画出插值曲线
# ynew = f(xnew)
# # -3.75 位置的点
# pl.scatter(-3.75, f(-3.75), color='m')
# pl.text(-3.75, f(-3.75), 'interpolate')
# # 0.25 位置的点
# pl.scatter(0.25, f(0.25), color='g')
# pl.text(0.25, f(0.25), 'interpolate')
# print("{} error:{}".format("linear", mean_squared_error(tag_y, ynew)))
# pl.plot(xnew, ynew, label=str("linear"))  # Label 用来显示图例
# #  目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  #  显示图例的位置
# pl.show()

# 3 次样条插值，物体 1，10 个点
```

```
# x_u = x_1
# y = y_1
# target = target1
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# # 用于估计预测值和准确值之间的误差
# tag_y = target(xnew)
# # 画出曲线
# pl.plot(x_u, y, "ro")
# pl.title("cubic spline")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f4 = interpolate.splrep(x_u, y)
# ynew4 = interpolate.splev(xnew, f4)
# #-3.75 的点
# pl.scatter(-3.75, interpolate.splev(-3.75, f4), color='m')
# pl.text(-3.75, interpolate.splev(-3.75, f4), 'interpolate')
# #0.25 的点
# pl.scatter(0.25, interpolate.splev(0.25, f4), color='g')
# pl.text(0.25, interpolate.splev(0.25, f4), 'interpolate')
# pl.plot(xnew, ynew4, label=str("cubic spline"))  # Label 用来显示图例
# print("{} error:{}".format('cubic spline', mean_squared_error(tag_y, ynew4)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  # 显示图例的位置
# pl.show()

# 3 次样条插值，物体 1，12 个点
# x_u = x_2
# y = y_2
# target = target1
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# # 用于估计预测值和准确值之间的误差
# tag_y = target(xnew)
# # 画出曲线
# pl.plot(x_u, y, "ro")
# pl.title("cubic spline")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
```

```
# f4 = interpolate.splrep(x_u, y)
# ynew4 = interpolate.splev(xnew, f4)
# # -3.75
# pl.scatter(-3.75, interpolate.splev(-3.75, f4), color='m')
# pl.text(-3.75, interpolate.splev(-3.75, f4), 'interpolate')
# # 0.25
# pl.scatter(0.25, interpolate.splev(0.25, f4), color='g')
# pl.text(0.25, interpolate.splev(0.25, f4), 'interpolate')
# pl.plot(xnew, ynew4, label=str("cubic spline"))  # Label 用来显示图例
# print("{} error:{}".format('cubic spline', mean_squared_error(tag_y, ynew4)))
# #  目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  #  显示图例的位置
# pl.show()

# 3 次样条插值，物体 1，14 个点
# x_u = x_3
# y = y_3
# target = target1
# #  用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# #  用于估计预测值和准确值之间的误差
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("cubic spline")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f4 = interpolate.splrep(x_u, y)
# ynew4 = interpolate.splev(xnew, f4)
# pl.scatter(-3.75, interpolate.splev(-3.75, f4), color='m')
# pl.text(-3.75, interpolate.splev(-3.75, f4), 'interpolate')
# pl.scatter(0.25, interpolate.splev(0.25, f4), color='g')
# pl.text(0.25, interpolate.splev(0.25, f4), 'interpolate')
# pl.plot(xnew, ynew4, label=str("cubic spline"))  # Label 用来显示图例
# print("{} error:{}".format('cubic spline', mean_squared_error(tag_y, ynew4)))
# #  目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  #  显示图例的位置
```

```
# pl.show()

# 3 次样条插值，物体 1，所有点
# x_u = x
# y = y1
# target = target1
# #  用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# #  用于估计预测值和准确值之间的误差
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("cubic spline")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f4 = interpolate.splrep(x_u, y)
# ynew4 = interpolate.splev(xnew, f4)
# pl.scatter(-3.75, interpolate.splev(-3.75, f4), color='m')
# pl.text(-3.75, interpolate.splev(-3.75, f4), 'interpolate')
# pl.scatter(0.25, interpolate.splev(0.25, f4), color='g')
# pl.text(0.25, interpolate.splev(0.25, f4), 'interpolate')
# pl.plot(xnew, ynew4, label=str("cubic spline"))  # Label 用来显示图例
# print("{} error:{}".format('cubic spline', mean_squared_error(tag_y, ynew4)))
# #  目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  #  显示图例的位置
# pl.show()

# 3 次样条插值，物体 2，10 个点
# x_u = x_1
# y = z_1
# target = target2
# #  用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# # #  用于估计预测值和准确值之间的误差
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("cubic spline")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f4 = interpolate.splrep(x_u, y)
```

```
# ynew4 = interpolate.splev(xnew, f4)
# # # -3.75
# pl.scatter(-3.75, interpolate.splev(-3.75, f4), color='m')
# pl.text(-3.75, interpolate.splev(-3.75, f4), 'interpolate')
# pl.scatter(0.25, interpolate.splev(0.25, f4), color='g')
# pl.text(0.25, interpolate.splev(0.25, f4), 'interpolate')
# pl.plot(xnew, ynew4, label=str("cubic spline"))  # Label 用来显示图例
# print("{} error:{}".format('cubic spline', mean_squared_error(tag_y, ynew4)))
# #  目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  #  显示图例的位置
# pl.show()

# 3 次样条插值，物体 2，12 个点
# x_u = x_2
# y = z_2
# target = target2
# #  用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("cubic spline")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f4 = interpolate.splrep(x_u, y)
# ynew4 = interpolate.splev(xnew, f4)
# pl.scatter(-3.75, interpolate.splev(-3.75, f4), color='m')
# pl.text(-3.75, interpolate.splev(-3.75, f4), 'interpolate')
# pl.scatter(0.25, interpolate.splev(0.25, f4), color='g')
# pl.text(0.25, interpolate.splev(0.25, f4), 'interpolate')
# pl.plot(xnew, ynew4, label=str("cubic spline"))  # Label 用来显示图例
# print("{} error:{}".format('cubic spline', mean_squared_error(tag_y, ynew4)))
# #  目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  #  显示图例的位置
# pl.show()

# 3 次样条插值，物体 2，14 个点
```

```
# x_u = x_3
# y = z_3
# target = target2
# #  用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("cubic spline")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f4 = interpolate.splrep(x_u, y)
# ynew4 = interpolate.splev(xnew, f4)
# pl.scatter(-3.75, interpolate.splev(-3.75, f4), color='m')
# pl.text(-3.75, interpolate.splev(-3.75, f4), 'interpolate')
# pl.scatter(0.25, interpolate.splev(0.25, f4), color='g')
# pl.text(0.25, interpolate.splev(0.25, f4), 'interpolate')
# pl.plot(xnew, ynew4, label=str("cubic spline"))  # Label 用来显示图例
# print("{} error:{}".format('cubic spline', mean_squared_error(tag_y, ynew4)))
# #  目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  #  显示图例的位置
# pl.show()

# 3 次样条插值，物体 2，所有点
# x_u = x
# y = y2
# target = target2
# #  用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# #  用所用的点画线
# pl.plot(x_u, y, "ro")
# pl.title("cubic spline")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f4 = interpolate.splrep(x_u, y)
# ynew4 = interpolate.splev(xnew, f4)
# pl.scatter(-3.75, interpolate.splev(-3.75, f4), color='m')
# pl.text(-3.75, interpolate.splev(-3.75, f4), 'interpolate')
# pl.scatter(0.25, interpolate.splev(0.25, f4), color='g')
```

```
# pl.text(0.25, interpolate.splev(0.25, f4), 'interpolate')
# pl.plot(xnew, ynew4, label=str("cubic spline"))  # Label 用来显示图例
# print("{} error:{}".format('cubic spline', mean_squared_error(tag_y, ynew4)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="upper right")  # 显示图例的位置
# pl.show()

# # 拉格朗日，物体 2，所有点
# x_u = x
# y = y2
# target = target2
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("lagrange")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f5 = interpolate.lagrange(x_u, y)
# ynew5 = f5(xnew)
# pl.scatter(-3.75, f5(-3.75))
# pl.text(-3.75, f5(-3.75), 'interpolate')
# pl.scatter(0.25, f5(0.25))
# pl.text(0.25, f5(0.25), 'interpolate')
# pl.plot(xnew, ynew5, label="Lagrange")
# print("{} error:{}".format('Lagrange', mean_squared_error(tag_y, ynew5)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="lower right")  # 显示图例的位置
# pl.show()

# # 拉格朗日，物体 2，10 个点
# x_u = x_1
# y = z_1
# target = target2
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
```

```
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("lagrange")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f5 = interpolate.lagrange(x_u, y)
# ynew5 = f5(xnew)
# pl.scatter(-3.75, f5(-3.75))
# pl.text(-3.75, f5(-3.75), 'interpolate')
# pl.scatter(0.25, f5(0.25))
# pl.text(0.25, f5(0.25), 'interpolate')
# pl.plot(xnew, ynew5, label="Lagrange")
# print("{} error:{}".format('Lagrange', mean_squared_error(tag_y, ynew5)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="lower right")  # 显示图例的位置
# pl.show()
#
# # # 拉格朗日，物体 2，12 个点
# x_u = x_2
# y = z_2
# target = target2
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("lagrange")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f5 = interpolate.lagrange(x_u, y)
# ynew5 = f5(xnew)
# pl.scatter(-3.75, f5(-3.75))
# pl.text(-3.75, f5(-3.75), 'interpolate')
# pl.scatter(0.25, f5(0.25))
# pl.text(0.25, f5(0.25), 'interpolate')
# pl.plot(xnew, ynew5, label="Lagrange")
# print("{} error:{}".format('Lagrange', mean_squared_error(tag_y, ynew5)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
```

```
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="lower right")  # 显示图例的位置
# pl.show()
#
# # # 拉格朗日，物体 2，14 个点
# x_u = x_3
# y = z_3
# target = target2
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("lagrange")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f5 = interpolate.lagrange(x_u, y)
# ynew5 = f5(xnew)
# pl.scatter(-3.75, f5(-3.75))
# pl.text(-3.75, f5(-3.75), 'interpolate')
# pl.scatter(0.25, f5(0.25))
# pl.text(0.25, f5(0.25), 'interpolate')
# pl.plot(xnew, ynew5, label="Lagrange")
# print("{} error:{}".format('Lagrange', mean_squared_error(tag_y, ynew5)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="lower right")  # 显示图例的位置
# pl.show()
#
# # # 拉格朗日，物体 1，所有点
# x_u = x
# y = y1
# target = target1
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("lagrange")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f5 = interpolate.lagrange(x_u, y)
```

```
# ynew5 = f5(xnew)
# pl.scatter(-3.75, f5(-3.75))
# pl.text(-3.75, f5(-3.75), 'interpolate')
# pl.scatter(0.25, f5(0.25))
# pl.text(0.25, f5(0.25), 'interpolate')
# pl.plot(xnew, ynew5, label="Lagrange")
# print("{} error:{}".format('Lagrange', mean_squared_error(tag_y, ynew5)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="lower right")  # 显示图例的位置
# pl.show()
#
# # # 拉格朗日，物体 1，10 个点
# x_u = x_1
# y = y_1
# target = target1
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("lagrange")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f5 = interpolate.lagrange(x_u, y)
# ynew5 = f5(xnew)
# pl.scatter(-3.75, f5(-3.75))
# pl.text(-3.75, f5(-3.75), 'interpolate')
# pl.scatter(0.25, f5(0.25))
# pl.text(0.25, f5(0.25), 'interpolate')
# pl.plot(xnew, ynew5, label="Lagrange")
# print("{} error:{}".format('Lagrange', mean_squared_error(tag_y, ynew5)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="lower right")  # 显示图例的位置
# pl.show()
#
# # # 拉格朗日，物体 1，12 个点
# x_u = x_2
```

```
# y = y_2
# target = target1
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("lagrange")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f5 = interpolate.lagrange(x_u, y)
# ynew5 = f5(xnew)
# pl.scatter(-3.75, f5(-3.75))
# pl.text(-3.75, f5(-3.75), 'interpolate')
# pl.scatter(0.25, f5(0.25))
# pl.text(0.25, f5(0.25), 'interpolate')
# pl.plot(xnew, ynew5, label="Lagrange")
# print("{} error:{}".format('Lagrange', mean_squared_error(tag_y, ynew5)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="lower right")  # 显示图例的位置
# pl.show()
#
# # 拉格朗日，物体 1，14 个点
# x_u = x_3
# y = y_3
# target = target1
# # 用于画出插值曲线
# xnew = np.linspace(x_u[0], x_u[-1], 101)
# tag_y = target(xnew)
# pl.plot(x_u, y, "ro")
# pl.title("lagrange")
# pl.xlabel('x 坐标')
# pl.ylabel('y 坐标')
# f5 = interpolate.lagrange(x_u, y)
# ynew5 = f5(xnew)
# pl.scatter(-3.75, f5(-3.75))
# pl.text(-3.75, f5(-3.75), 'interpolate')
# pl.scatter(0.25, f5(0.25))
# pl.text(0.25, f5(0.25), 'interpolate')
# pl.plot(xnew, ynew5, label="Lagrange")
```

```
# print("{} error:{}".format('Lagrange', mean_squared_error(tag_y, ynew5)))
# # 目标
# x_target = np.linspace(-5, 5, 1000)
# y_target = target(x_target)
# pl.plot(x_target, y_target, label="target")
# pl.legend(loc="lower right")  # 显示图例的位置
# pl.show()
```

6 最小二乘法的实验与分析

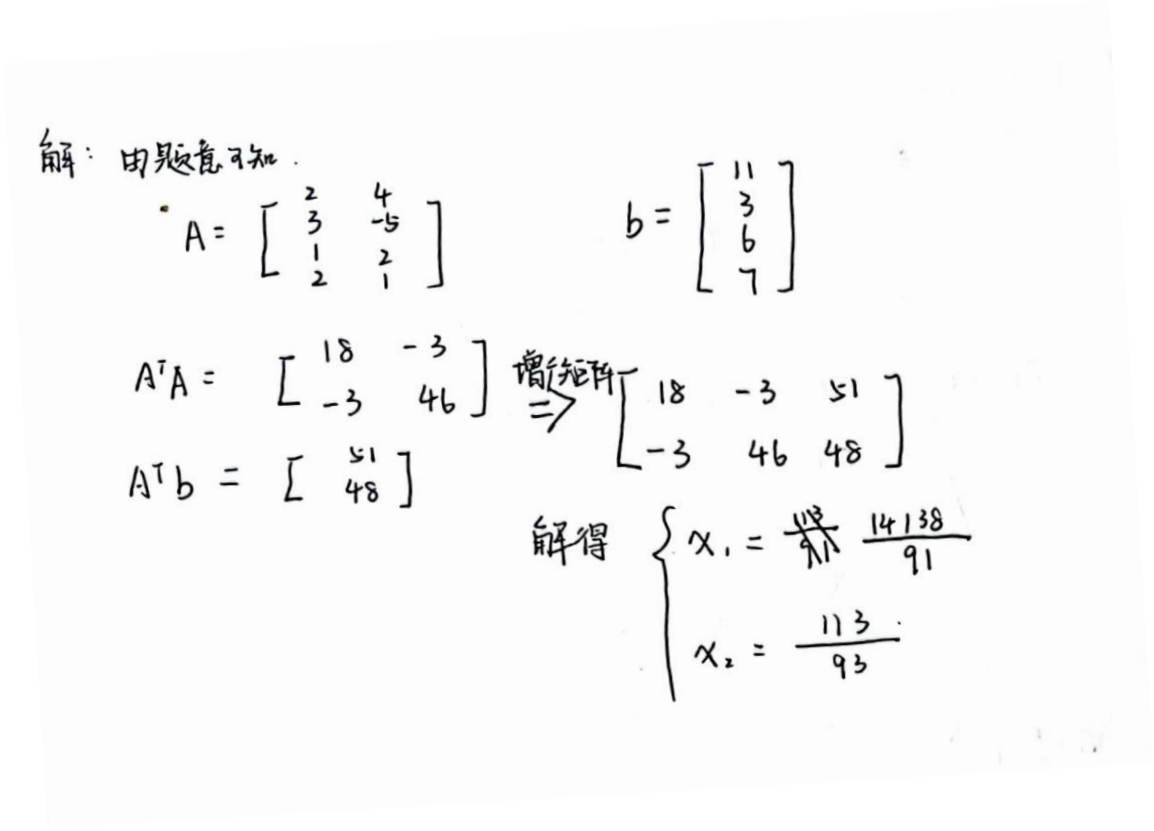
6.1 计算题

6.1.1 题目 1

求超定方程组：

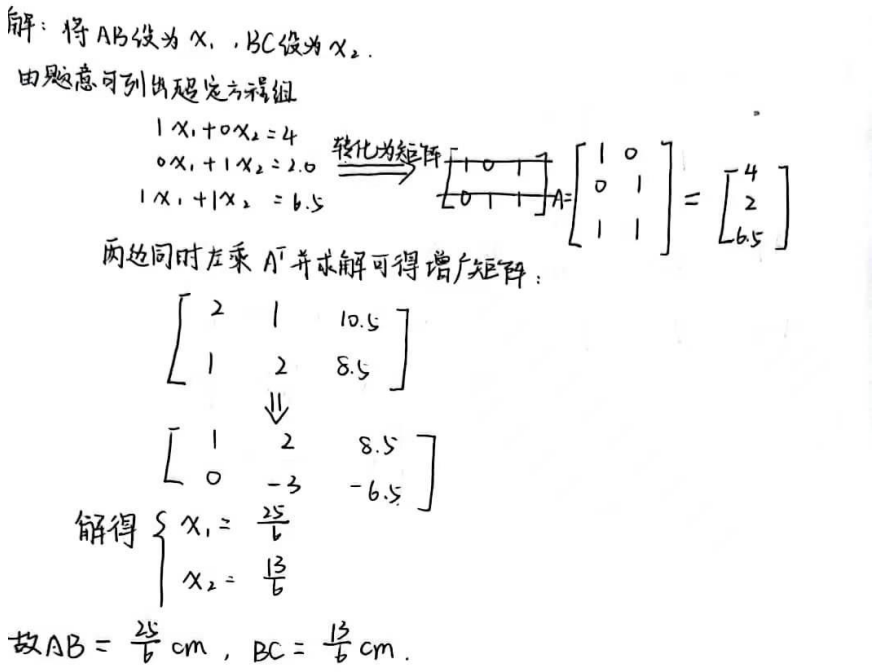
$$\begin{cases} 2x_1 + 4x_2 = 11 \\ 3x_1 - 5x_2 = 3 \\ x_1 + 2x_2 = 6 \\ 2x_1 + x_2 = 7 \end{cases}$$

的最小二乘解。



6.1.2 题目 2

某人粗略地测量到线段 *AB* 的长度为 4.0 厘米，线段 *BC* 的长度为 2.0 厘米，线段 *AC*＝*AB*+*BC* 的长度为 6.5 厘米。试利用最小二乘原理合理地确定线段 *AB* 和 *BC* 的长度。



6.1.3 题目 3

给定如下观察数据

x	1	2	4	5
y	0.33	0.40	0.44	0.45

求一个形如 $y=\dfrac{x}{c_0x+c_1}$ 的函数，使得函数对上述数据最小二乘拟合。

依题意得

$$y=\frac{x}{c_0x+c_1}\Rightarrow\frac{1}{y}=c_0+c_1\frac{1}{x}$$

可列出正规方程

$$\begin{bmatrix}\sum_{i=1}^{5}1 & \sum_{i=1}^{5}\frac{1}{x_i}\\ \sum_{i=1}^{5}\frac{1}{x_i} & \sum_{i=1}^{5}\frac{1}{x_i^2}\end{bmatrix}\begin{bmatrix}c_0\\ c_1\end{bmatrix}=\begin{bmatrix}\sum_{i=1}^{5}\frac{1}{y_i}\\ \sum_{i=1}^{5}\frac{1}{y_i}\frac{1}{x_i}\end{bmatrix}$$

代入数据可得

$$\begin{bmatrix}4 & \frac{39}{20}\\ \frac{39}{20} & \frac{541}{400}\end{bmatrix}\begin{bmatrix}c_0\\ c_1\end{bmatrix}=\begin{bmatrix}\frac{65505}{6534}\\ \frac{1572}{297}\end{bmatrix}$$

解得 $c_0\approx 2.5922$　　$c_1\approx -0.17614$

$$\therefore y=\frac{x}{2.5922x-0.4761}$$

6.2 综合实验：石油产量预测

6.2.1 实验题目

世界石油产量以每天百万桶计，如表 1 所示，求最佳最小二乘法数值估计：

表 1 世界石油产量

年	桶/天（$\times 10^6$）	年	桶/天（$\times 10^6$）
1994	67.052	1999	72.063
1995	68.008	2000	74.699
1996	69.803	2001	74.487
1997	72.024	2002	74.065
1998	73.400	2003	76.777

【实验要求】

1. 分别分析采用(a)直线，(b)抛物线，(c)立方曲线拟合 10 个数据点的结果，写出拟合后的具体函数表达式。
2. 进行如表 1 的残差分析。就残差分析结果而言，哪一种拟合最好的代表了这些数据？
3. 利用上面的每一种拟合来估计 2010 年的石油生产水平，讨论结果。

6.2.2 算法介绍

1. 采用(a)直线，(b)抛物线，(c)立方曲线拟合

   (a) $y = kx + b$

   (b) $y = k_1 x^2 + k_2 x + b$

   (c) $y = k_1 x^3 + k_2 x^2 + k_3 x + b$

2. 采用 leastsq 函数

   假设“逼近”规律的近似函数为 $y = f(x)$ 即有

$$y_i^* = f(x_i) \quad i = 1, 2, ..., m$$

它与观测值 y_i 之差 $\delta_i = y_i^* - y_i = f(x_i) - y_i$ 称为残差。

残差大大小可以作为衡量近似函数好坏的标准。按照使残差的平方和 $\sum \delta_i^2$ 最小的规则求得近似函数 $y = f(x)$ 的方法称为最佳平方逼近，也称为曲线拟合的最小二乘法。

6.2.3 实验结果及分析

2010 年直线，抛物线，立方曲线三种拟合方式的预测结果为：

[82.7880, 65.3525, 68.7086]

各个曲线拟合的表达式为：

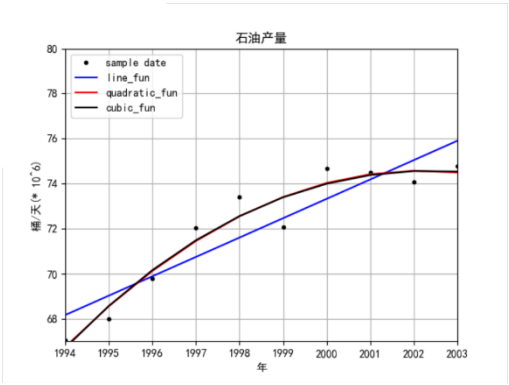
- 直线： $y = 0.8603x + 68.1636$
- 抛物线： $y = -0.1178x^2 + 1.9205x + 66.7500$
- 立方曲线： $y = 1.89 \times 10^{-3} x^3 + (-0.1434)x^2 + 2.0079x + 66.7022$

实验结果的二范数的平方、残差的标准差和绝对值的均值如表 2 所示，拟合效果如图 6.1 所示：

表 6.1 二范数的平方、残差的标准差和绝对值的均值

多项式类型	二范数的平方	残差的标准差	残差绝对值的均值
直线	11.4296	1.0691	0.9441
抛物线	4.1018	0.6404	0.5431
立方曲线	4.0906	0.6939	0.5483

图 6.1 三种不同多项式类型的结果拟合图



【结果分析】

根据表 6.1 可得出，立方曲线拟合是效果最佳的拟合方式，由图 6.1 可知，抛物线拟合与立方曲线拟合效果十分接近，线性拟合效果较差，所得出的预测结果也不具有参考价值。

6.2.4 附录：源代码

```
把源代码贴于此处
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
from fitting import *
from scipy.optimize import leastsq

mpl.rcParams['font.sans-serif'] = ['SimHei']   # 指定默认字体
mpl.rcParams['axes.unicode_minus'] = False   # 解决负号‘-’无法显示的问题
x = np.arange(0, 10, 1, dtype='float')
x_plt = np.arange(1994, 2004, 1, dtype='float')
y = np.array([67.052, 68.008, 69.803, 72.024, 73.400, 72.063, 74.669, 74.487,
74.065, 74.777])

def line_fun(s, x):
    k1, b = s
    return k1 * x + b


def quadratic_fun(s, x):
    k1, k2, b = s
```

```
    return k1 * x ** 2 + k2 * x + b


def cubic_fun(s, x):
    k1, k2, k3, b = s
    return k1 * x ** 3 + k2 * x ** 2 + k3 * x + b


def fpower_fun(s, x):
    k1, k2, k3, k4, b = s
    return k1 * x ** 4 + k2 * x ** 3 + k3 * x ** 2 + k4 * x + b


def dist(a, fun, x, y):
    return fun(a, x) - y


plt.figure()
plt.title(u'石油产量')
plt.xlabel(u'年')
plt.ylabel(u'桶/天(* 10^6)')

plt.axis([1994, 2003, 67, 80])
plt.grid(True)
plt.plot(x_plt, y, 'k.')
# plt.show()

param1 = np.array([0, 0])
param2 = np.array([0, 0, 0])
param3 = np.array([0, 0, 0, 0])
funs = [line_fun, quadratic_fun, cubic_fun]
params = [param1, param2, param3]
colors = ['blue', 'red', 'black']
fun_name = ['line_fun', 'quadratic_fun', 'cubic_fun']
vars = []
for i, (func, param, color, name) in enumerate(zip(funs, params, colors,
fun_name)):
    var = leastsq(dist, param, args=(func, x, y))   # 求出残差平方和最小的待定
系数的值
    vars.append(var)
    plt.plot(x_plt, func(var[0], x), color)
    print('[%s]  二范数: %.4f, abs(bias): %.4f, bias-std: %.4f' % (name,
```

```
((y - func(var[0], x)) ** 2).sum(),

(y - func(var[0], x)).std(),

(abs(y - func(var[0], x))).mean())
                )
plt.legend(['sample date', 'line_fun', 'quadratic_fun', 'cubic_fun'], loc='upper left')

print(vars)
x_predict = 2010 - 1994 + 1
y_predict = {}
for i, func in enumerate(funs):
      y_predict[str(fun_name[i])] = funs[i](vars[i][0], x_predict)
print(y_predict)
plt.show()
```

7 数值积分、微分和常微分方程的数值解法

7.1 数值积分

7.1.1 题目 1

用梯形公式计算积分 $\int_0^1 x^2 \mathrm{d}x$。

解：

$$\int_0^1 x^2 dx = \frac{b-a}{2}[f(a)+f(b)] = \frac{1}{2}$$

7.1.2 题目 2

用辛普森公式计算积分 $\int_1^2 \sqrt{x}\mathrm{d}x$。

解：

1/3 辛普森公式计算：

$$C_0^{(2)} = \frac{1}{6}, C_1^{(2)} = \frac{2}{3}, C_2^{(2)} = \frac{1}{6}$$

$$\int_a^b \sqrt{x}\,dx \approx \frac{h}{3}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right], h=\frac{b-a}{2}$$

$$\int_1^2 \sqrt{x}\,dx \approx (2-1)\left(\frac{1}{6}\times\sqrt{1}+\frac{4}{6}\times\sqrt{1.5}+\frac{1}{6}\times\sqrt{2}\right)=1.218866$$

3/8 辛普森公式计算：

$$C_0^{(3)}=\frac{1}{8}, C_1^{(3)}=\frac{3}{8}, C_2^{(3)}=\frac{3}{8}, C_3^{(3)}=\frac{1}{8}$$

$$\int_a^b f(x)dx \approx \frac{3h}{8}\left[f(a)+3f(c)+3f(d)+f(b)\right], h=\frac{b-a}{3}$$

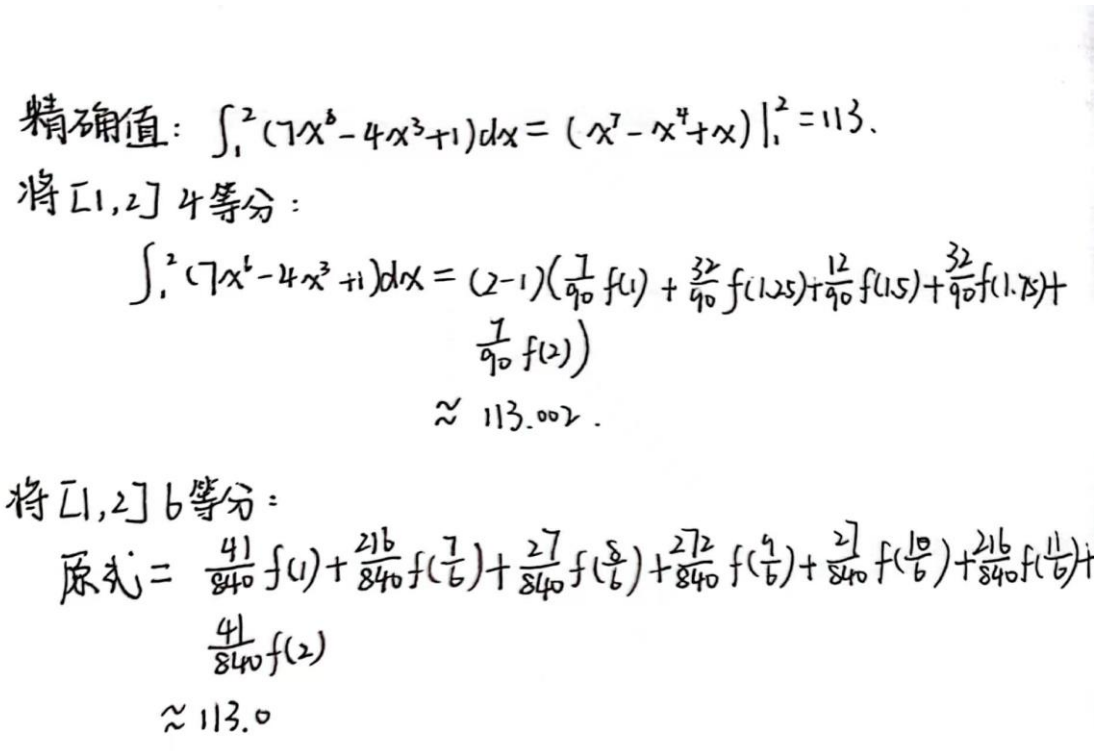
$$\int_1^2 \sqrt{x}\,dx \approx (2-1)\left(\frac{1}{8}\times\sqrt{1}+\frac{3}{8}\times\sqrt{\frac{4}{3}}+\frac{3}{8}\times\sqrt{\frac{5}{3}}+\frac{1}{8}\times\sqrt{2}\right)=1.218912$$

7.1.3 题目 3

分别用 4 次、6 次牛顿－柯特斯公式计算以下积分并与精确值作比较。

$$\int_1^2 (7x^6-4x^3+1)\mathrm{d}x$$

解：



7.2 数值微分

7.2.1 题目 1

用三点插值微分公式求各点的一阶和二阶导数，函数 $f(x)$ 由下表给出

表 7.1 三点函数值

x_i	1.0	1.1	1.2
$f(x_i)$	0.2500	0.2268	0.2066

解：

【算法介绍】

三点插值多项式求一阶导数步骤：

1. 已知函数等距的三个点，需要求函数在这三个点的导数值。
2. 求出经过三个点的二次多项式插值函数。
3. 以函数的导数代替原函数的导数。
4. 三点微分公式求一阶导数

$$\begin{cases} f''(1.0)=\dfrac{1}{0.04}\left[f(1.0)-2f(1.1)+(1.2)\right] \\ f''(1.1)\approx\dfrac{1}{0.04}\left[f(1.0)-2f(1.1)+(1.2)\right] \\ f''(1.2)=\dfrac{1}{0.04}\left[f(1.0)-2f(1.1)+(1.2)\right] \end{cases}$$

二阶导数公式：

$$f''(x)=\frac{1}{h^2}\left[f(x_0)-2f(x_1)+f(x_2)\right]$$

【求解】

三点公式求一阶导数：

$$\begin{cases} f'(1.0)=\dfrac{1}{0.2}\left[-3f(1.0)+4f(1.1)-f(1.2)\right] \\ f'(1.1)\approx\dfrac{1}{0.2}\left[-f(1.0)+f(1.1)\right] \\ f'(1.2)=\dfrac{1}{0.2}\left[f(1.0)-4f(1.1)+3f(1.2)\right] \end{cases}$$

三点公式求二阶导数：

$$\begin{cases} f''(1.0)=\dfrac{1}{0.04}\left[f(1.0)-2f(1.1)+(1.2)\right] \\ f''(1.1)\approx\dfrac{1}{0.04}\left[f(1.0)-2f(1.1)+(1.2)\right] \\ f''(1.2)=\dfrac{1}{0.04}\left[f(1.0)-2f(1.1)+(1.2)\right] \end{cases}$$

得到最终解：

表 7.2 三点插值微分公式求的一阶和二阶导数值

x	1.0	1.1	1.2
一阶	-0.247	-0.217	-0. 817
二阶	0.300	0.300	0.300

7.3 常微分方程的数值解法

7.3.1 题目 1

分别用欧拉方法、改进欧拉法求解初值问题

$$y' = \frac{t-y}{2}, \quad y(0)=1$$

在区间[0,3]上的数值解，取步长为 $h=0.5$，比较它们与准确解的误差（已知该初值问题的解析解为 $y(t)=3e^{-t/2}-2+t$）。

【算法介绍】

欧拉法公式：

$$y_{i+1} = y_i + hf(t_i, y_i)$$

改进欧拉法公式：

$$y_{i+1}^{(1)} = y_i + hf(t_i, y_i)$$

$$y_{i+1}^{(2)} = y_i + \frac{h}{2}\left[f(t_i, y_i) + f(t_{i+1}, y_{i+1}^{(1)}) \right]$$

表 7.3 区间[0,3]内 $y'=t-y/2$，$y(0)=1$ 不同步长欧拉方法比较

t_k	y_k				精确解 $y(t_k)$
	h=1	h=1/2	h=1/4	h=1/8	
0	1.0	1.0	1.0	1.0	1.0
0.125				0.9375	0.943239
0.25			0.875	0.886719	0.897491
0.375				0.846924	0.862087
0.50		0.75	0.769875	0.817429	0.836402
0.75			0.759766	0.786802	0.811868
1.00	0.5	0.6875	0.758545	0.790158	0.819592
1.50		0.765625	0.846836	0.882885	0.917100
2.00	0.75	0.949219	1.030827	1.068222	1.103638
2.50		1.211914	1.289227	1.325179	1.359514
3.00	1.375	1.533926	1.604252	1.637429	1.669390

【问题求解】

根据算法介绍和表 7.1 可得当步长为 0.5 时，数值解与精确解的误差，如表 7.2 所示。

表 7.4 h=0.5 的数值解与精确解的误差

t_k	h=1/2	精确解 $y(t_k)$	误差

0	1.0	1.0	0
0.125		0.943239	
0.25		0.897491	
0.375		0.862087	
0.50	0.75	0.836402	0.086402
0.75		0.811868	
1.00	0.6875	0.819592	0.132092
1.50	0.765625	0.917100	0.151475
2.00	0.949219	1.103638	0.154419
2.50	1.211914	1.359514	0.1476
3.00	1.533926	1.669390	0.135464

7.4 二阶龙格-库塔方法

7.4.1 题目描述

用二阶龙格-库塔方法求解初值问题

$$y' = -\frac{y}{t+y^2}, \quad y(0)=1$$

在 $t=1$ 上的数值解，取步长 $h=0.5$ 。

7.4.2 问题求解求解

表 7.5　二阶龙格-库塔方法求解

tk	二阶龙格-库塔方法
0.50	1.000000
1.00	0.802083
2.00	0. 528671
2.50	0. 354057
3.00	0. 253407
3.50	0. 152716
4.00	0. 125341

7.4.3 源代码

```
def rk2_method(f, t0, y0, h, num_steps):
    t = [0] * (num_steps + 1)
    y = [0] * (num_steps + 1)
    t[0] = t0
    y[0] = y0
```

```
    for i in range(num_steps):
        k1 = f(t[i], y[i])
        k2 = f(t[i] + h / 2, y[i] + h / 2 * k1)
        t[i + 1] = t[i] + h
        y[i + 1] = y[i] + h * k2
    return t, y

# 定义微分方程的右侧函数
def f(t, y):
    return -(y / t + y ** 2)

if __name__ == '__main__':
    # 设置初始条件和求解参数
    t0 = 0.5
    y0 = 1
    h = 0.5
    num_steps = 10
    # 使用二阶龙格-库塔方法求解微分方程
    t, y = rk2_method(f, t0, y0, h, num_steps)

    # 打印结果
    for i in range(len(t)):
        print(f"t = {t[i]:.2f}, y = {y[i]:.6f}")
```

8 总结与心得体会

8.1 总结

本文将经典数值计算中的问题：误差、非线性方程、线性方程组、插值法、最小二乘拟合、数值积分、数值微分和常微分方程共 8 个问题进行了介绍，并针对每一种问题，选取了该问题上的几个经典方法和题目通过 Python 编程、画图等方法进行了解答以及分析。

8.2 心得体会

经过一学期的 Python 数值分析课程，我在数学能力、Python 编程能力以及文档规范和排版能力等方面得到了显著提升。

首先，在数学能力方面，尽管我之前对自己的数学能力很有信心，但在课程开始时，我发现自己在数学推导方面有些力不从心。为了克服这个问题，我重新温习了数学建模课本，并结合课堂内容进行复习。这让我逐渐恢复了状

态，并且在原有的数学建模基础上学到了更多新知识。这不仅对我下次参加数学建模竞赛有很大帮助，也为我未来学术道路的发展提供了支持。

其次，在 Python 编程能力方面，之前我只是对基本的 Python 使用和调库有所了解，对于数值分析常用的库如 Numpy、Pandas 和 Matplotlib 只是略有涉猎。因此，在课程开始时，我发现自己在编程方面有些困难，经常遇到问题。于是我决心提升自己的 Python 编程能力，特别是数值分析能力。经过一个学期的学习，我现在更加熟练地进行数据分析和绘图工作，编程风格也更加规范。Python 作为一门强大的语言，掌握它对我的未来发展非常有益。

最后，在文档规范和排版能力方面，之前我在文档编写和排版方面相对一般，常常草率完成。然而，在这门课程中，我意识到文档排版的重要性，特别是在撰写论文等场景下。因此，我学习了 Word 的排版技巧，并咨询了熟悉排版的朋友和师兄师姐的意见。他们给了我很多宝贵的修改建议，并让我意识到了应该注意的细节。我相信在今后撰写实验报告和课程报告等文档时，我可以运用所学知识，完成任务得体。

除了上述能力的提升，我还在心态和眼界方面有所成长。在检查排版和数据等任务中，我学会了细心观察细节并且做事一丝不苟。当数据出现错误时，我学会了如何逻辑清晰地进行错误排查。在课程学习过程中，我还观摩学习了几位同学的优秀作业，从中获得了许多经验

参考文献：

[1] http://t.zoukankan.com/startover-p-3141676.html

[2] 吕同富，康兆敏，方秀男，等编著.数值计算方法[M].北京：清华大学出版社，2013.27-28

[3] 李军, 周鹏. GPS 定位误差原因研究[J]. 中国高新技术企业, 2015(31):2.

[4] 谭子明. 实验曲线多项式拟合时次数的统计推断[J]. 工程数学学报, 1987(2):106-109.

[5] 刘志彬. 秦九韶与《数书九章》[M]. 北京师范大学出版社, 1987.

[6] 戚兴龙. 二分法求解单变量非线性方程及其应用与实现[J]. 商情, 2009(33):1.

[7] 李成林. 不动点问题中使用 Aitken 算法的讨论[J]. 工程技术(文摘版), 2016(2):00257-00257.

[8] Guo X , Qiu-Yue L I , Wang-Li X U . Acceleration of the EM Algorithm Using the Vector Aitken Method and Its Steffensen Form[J]. 应用数学学报：英文版, 2017, 33(1):8.